INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

CAKE
Hybrides Gruppen-schlüsselmanagementprotokoll für RIOT OS

Edgar Goetzendorff

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

CAKE
Hybrides Gruppenschlüsselmanagementprotokoll für RIOT OS

Edgar Goetzendorff

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller
Betreuer:	Dr. Nils gentschen Felde Tobias Guggemos Dr. Peter Hillmann (Universität Bundeswehr) Klement Streit (Universität Bundeswehr)
Abgabetermin:	6. April 2018

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 6. April 2018

. . . . . .
(Unterschrift des Kandidaten)

Vorwort

Diese Bachelorarbeit ist das Ergebnis gemeinschaftlicher Arbeit von Edgar Goetzendorff und Mehdi Yosofie. So ist auch der während dieser Arbeit erstellte Quellcode aus gemeinsamer Mitarbeit entstanden. In dieser Arbeit sind die Kapitel, die jeweils vom anderen Mitwirkenden sind, eindeutig als Zitat markiert.

Abstract

Gruppenorientierte Kommunikation in der Informationstechnologie gewinnt zunehmend an Bedeutung für das tägliche Leben, die Wirtschaft oder die nationale Sicherheit. Eine sichere Gruppenkommunikation erfordert die Gewährleistung von gewissen Sicherheitsdiensten. Diese beinhalten unter anderem Vertraulichkeit, Authentizität und Integrität. Eine weitere Anforderung ist die effiziente Erstellung und Verteilung von geeignetem Schlüsselmaterial an die einer Gruppe zugehörigen Nutzer. Das Schlüsselmanagementprotokoll CAKE wurde dahingehend konzipiert, die benötigte Rechenleistung für die Generierung der Schlüssel, sowie die Anzahl der für die Verteilung der Schlüssel notwendigen Nachrichten, im Vergleich mit bereits etablierten Verfahren wie GKMP oder LKH, so gering wie möglich zu halten. Vor Im Laufe dieser Bachelorarbeit wurden die Software eines Clients sowie einen Key Server / Group Controllers für das Betriebssystem RIOT OS implementiert.

Abkürzungsverzeichnis

AI	Authorisierte Instanz
CAKE	Central Authorized Key Extension
CRT	Chinese Remainder Theorem
FIFO	First-In-First-Out
G-IKEv2	Group Internet Key Exchange version 2
GCC	GNU Compiler Collection
GCKS	Group Controller / Key Server
GC	Group Controller
GDOI	Group Domain of Interpretation
GKEK	Group Key Encryption Key
GKMP	Group Key Management Protocol
GKP	Group Key Paket
GMP	GNU Multiple Precision Arithmetic Library
GM	Gruppenmitglied
GTEK	Group Traffic Encryption Key
IdD	Internet der Dinge
IKEv2	Internet Key Exchange version 2
IKE	Internet Key Exchange
IPC	Inter Process Communication
ISAKMP	Internet Security Association and Key Management Prokoll
KEK	Key Encryption Key
LKH	Logical Key Hierarchy
MCU	Microcontroller Unit
PSK	Pre Shared Key
SA	Security Association
SL	Secure Lock

Inhaltsverzeichnis

1 Einleitung

Immer mehr Geräte werden vernetzt, schließen sich zu Gruppen zusammen und tauschen teils sensible Daten untereinander aus. Die Gewährleistung von verschiedenen Sicherheitsdienste wie Vertrautlichkeit, Authentizität und Datenintegrität sowie die sichere und effiziente Verschlüsselung der hierbei versendeten Daten ist unabdingbar. Vor allem bei einem Netz aus kompakten mobilen Endgeräten sind zwei Faktoren von entscheidender Bedeutung: zum Einen besitzen die Geräte keine hohe Rechenleistung, zum Anderen ist die Übertragungsbandbreite innerhalb des Netzes, im Gegensatz zu einem drahtgebundenen Netz, stark eingeschränkt. Für die Berechnung und Verteilung der notwendigen Schlüssel innerhalb einer Gruppe wurden bereits mehrere Verfahren entwickelt, welche unter anderem folgende Sicherheitsanforderungen erfüllen.

- **Backward Secrecy** Durch die *Backward Secrecy* wird erreicht, dass neue Teilnehmer einer Gruppe nicht im Stande sind, im Vorfeld erhaltene Nachrichten zu entschlüsseln.
- **Forward Secrecy** Verlässt ein Teilnehmer die Gruppe, wird durch die *Forward Secrecy* verhindert, dass dieser weiterhin, im Anschluss abgefangene, Nachrichten entschlüsseln kann.
- **Schlüsselunabhängigkeit** Die Kenntnis eines Schlüssels darf keine Schlussfolgerung über andere Schlüssel geben.

Diese Verfahren können in zentralisierte Protokolle, dezentralisierte Protokolle sowie in Protokolle mit verteilter Schlüsselvereinbarung gegliedert werden, wobei in dieser Arbeit nur die zentralisierten Protokolle betrachtet werden.

Bei zentralisierten Protokollen, wie etwa dem Group Key Management Protocol (GKMP) und dem Secure Lock (SL) Protokoll übernimmt eine zentrale Instanz, welche in der Regel mit mehr Rechenleistung sowie Speicherkapazität ausgestattet ist, die Berechnung und Verteilung des Schlüsselmaterials. Diese Arbeit beschäftigt sich mit der Implementierung eines weiteren zentralisierten Protokolls: CAKE.

CAKE (Central Authorized Key Extension) ist ein hybrides Verfahren, welches Merkmale von Group Key Management Protocol (GKMP), Logical Key Hierarchy (LKH) und Secure Lock (SL) miteinander vereint. Wie bei GKMP gibt es unter anderem einen Group Traffic Encryption Key (GTEK) sowie einen Group Key Encryption Key (GKEK). Die Schlüssel werden von der zentralen Instanz in einem Baum verwaltet, dieser ist allerdings kein binärer Baum wie bei LKH, sondern *ternär*. Für die Berechnung der Schlüssel wird, wie auch bei LKH und SL, das Chinese Remainder Theorem (CRT) verwendet.

Ziel dieser Arbeit ist die Erläuterung des Wissenschaftsgebiets, eine Implementierung des CAKE-Protokolls als klassisches Client-Server-Modell für RIOT-OS, sowie die Beschreibung des Programmflusses und der verwendeten Datenstrukturen.

Weiterer Aufbau dieser Arbeit

——— Der folgende Text ist ein direktes Zitat von [22]. ———

apitel 2 gibt einen Überblick über bereits bestehende Gruppenschlüsselprotokolle und Schlüsselmanagementverfahren sowie über das Betriebssystem RIOT OS und warum es für den Einsatz auf kompakten, mobilen Geräten geeignet ist. In Kapitel 3 wird das Konzept und die grundlegende Architektur von CAKE erklärt.

Auf den vorherigen Kapiteln aufbauend werden in Kapitel 4 das Protokolldesign sowie die einzelnen Rollen im CAKE System ausführlich beschrieben. Anschließend wird in Kapitel 5 auf die Implementierung und die damit verbundenen Datenstrukturen eingegangen. In Kapitel 6 folgt eine Bewertung des konzipierten Protokolls und mit Kapitel 7 endet diese Arbeit mit einer Zusammenfassung und einem Ausblick.

——— Ende des Zitats. ———

2 Grundlagen

——— Der folgende Text ist ein direktes Zitat von [22]. ———

n diesem Kapitel werden Gruppenschlüsselprotokolle wie GKMP, Group Domain of Interpretation (GDOI) und Group Internet Key Exchange version 2 (G-IKEv2) beschrieben und die Schlüsselmanagementverfahren LKH und SL erklärt, auf denen CAKE basiert.

Die vorliegende Arbeit integriert Teile des Group-Internet Key Exchange (IKE)v2 Protokolls, um einen sicheren Kanal zwischen dem CAKE GCKS und den Clients herzustellen und Gruppeninformationen wie den Gruppenschlüssel selber sicher zu übertragen. Eine initiale Gruppenerzeugung findet in CAKE mithilfe des broadcastbasierten SL statt. Bei einem Gruppeneintritt (Join Operation) im CAKE System wird bei der Verteilung des neuen Gruppenschlüssels wie beim GKMP vorgegangen. Bei einem Austritt (Leave Operation) wird mit der Idee des LKH und die Berechnungen des SL eine ternäre Baumstruktur verwaltet, damit der Nachrichtenaufwand gering bleibt.

——— Ende des Zitats. ———

2.1 Szenario

——— Der folgende Text ist ein direktes Zitat von [22]. ———

ls Anwendungsfall für die Verwendung einer gesicherten Gruppenkommunikation kann eine militärische Situation beatrachtet werden, an der die typischen Gruppenoperationen abzuleiten sind.

Im Falle eines Einsatzes werden einige Soldaten zu einer Einheit gerufen und bilden gemeinsam eine Gruppe. Zur Unterstützung der Gruppe wird kurz darauf ein weiterer Soldat hinzugesandt. Das entspricht dem Einzeleintritt. Auf dem Weg zum Einsatzgebiet wird die Truppe zudem von einer weiteren Kolonne bekräftigt. Für die weitere Kommunikation müssen beide Gruppe miteinander verschmolzen werden. Am Einsatzgebiet angekommen, wird ein Späher entsandt, wärend die restlichen Soldaten weiterhin eine geschlossene Einheit bilden. Der Späher wurde hierbei mithilfe einer Einzelaustritt-Operation aus der Gruppe entfernt, und kann somit keine weiteren Gruppen-Nachrichten entschlüsseln. Nach vollendetem Einsatz lösen sich die beiden Teilgruppen und marschieren jeweils zu ihren Lagern. Dies entspricht der Gruppentrennung.

An diesem Szenario werden die verschiedenen Gruppenoperationen ersichtlich, die ein Gruppenschlüsselmanagementprotokoll bereitstellen sollte. Die Soldaten tauschen während der ganzen Kommunikation per Funk ihre Nachrichten aus. Währenddessen darf ihre Kommunikation durch feindliche Angriffe nicht gelesen werden. Von daher ist eine verschlüsselte Übertragung der Nachrichten enorm wichtig, denn die sensiblen Daten dürfen nur im Kreise der authorisierten Armeeangehörigen bleiben.

——— Ende des Zitats. ———

2.2 RIOT OS

An eingebettete Systeme werden eine Vielzahl an Ansprüchen in Bezug auf Zuverlässigkeit, Echtzeitverhalten sowie Leistung gestellt. Ein geeignetes Betriebssystem für diese Art von Systemen muss demzufolge diesen Ansprüchen gerecht werden. Zudem sollte das Betriebssystem in der Lage sein, auf verschiedenster Hardware zu laufen, da es sich bei den eingesetzten Geräten um rechenschwache Microcontroller Units (MCU) bis hin zu stärkeren Geräten mit modernen, effizienten 32-bit Prozessoren handeln kann.

Grundsätzlich kann ein Betriebssystem durch drei Schlüsseleigenschaften klassifiziert werden: (i) Die Struktur des Kernels, (ii) die Funktionsweise des Schedulers, (iii) das Programmiermodell.

(i) Die Struktur des Kernels Der Kernel eines Betriebssystem kann monolitisch oder geschichtet aufgebaut sein oder eine Mikrokernel-Architektur implementieren. Ein monolithischer Kernel ist zwar unkompliziert zu designen, endet aber meist in einer schwer nachvollziehbaren, komplexen Struktur. Durch einen geschichteten Kernel kommt ein hierarchisches Modell hinzu, da als Entwickler entschieden werden muss, wie stark die Trennung zwischen Kernel- und User-Space erfolgt. Bei der Mikrokernel-Architektur wird eine noch höhere Modularität erreicht. Fehler in einzelnen Komponenten bringen nicht das gesamte System zum Absturz; hierdurch wird eine höhere Zuverlässigkeit gewährleistet.

(ii) Funktionsweise des Schedulers Die Wahl der Scheduling-Strategy ist entscheidend für die Unterstützung von Echtzeitverhalten sowie Prioritisierung von Prozessen und Nutzerinteraktionen. Strategien wie First-In-First-Out (FIFO) können beispielsweise den Prozessen keine faire Zuteilung von Resourcen gewährleisten.

(iii) Programmiermodell Ebenfalls von großer Bedeutung ist die Wahl des Programmiermodells. Die Prozesse können entweder alle in ein und dem selben Kontext verarbeitet werden, ohne Segmentierung des Speichers und Zuteilung bestimmter Bereiche des gleichen, oder jeder in seinem eigenen Thread, mit eigenem Speicherstapel.

Im Folgenden werden die zwei vorherrschenden Betriebssysteme für das Internet der Dinge (IdD), Contiki [4] und Tiny OS [12], sowie das vollwertige Betriebssystem Linux verglichen und dargestellt, welche Vorteile RIOT gegenüber diesen bringt.

Tiny OS und Linux implementiere beide einen monolitischen Kernel, während der von Kernel Contiki modular aufgebaut ist, und eher einem geschichteten System entspricht. Das Scheduling von Contiki und Tiny OS erfolgt nach dem FIFO-Prinzip, und ist somit auf die Abarbeitung von kleinen Prozessen optimisiert.

Der Scheduler von Linux kommt der Completely Fair Schedular (CFS) zum Einsatz. Durch die Verwendung eines Rot-Schwarz-Baumes wird eine faire Zuordnung der Prozessorzeit gewährleistet. Das Programmiermodell von Contiki und Tiny OS entspricht einem ereignisbasierten Modell, wobei alle Tasks im selben Kontext abgearbeitet werden. Volles Multi-Threading wird nicht erreicht, jedoch bieten beide Betriebssysteme eine teilweise Multi-Threading Unterstützung.

Die Benutzerfreundlichkeit wird bei Contiki und Tiny OS dadurch beeinträchtigt, das Contiki nur eine Teilmenge der Programmiersprache C zur Verfügung stellt, wodurch bestimmte Schlüsselwörter nicht verwendet werden können, während Tiny OS in einem eigenen C-Dialekt geschrieben wurde (nesC). Linux unterstützt sowohl C als auch C++.

RIOT wurde dahingehend konzipiert, die Unterschiede zwischen klassischen, vollwertigen Betriebssystemen und solchen, die auf eingebettete Systeme ausgerichtet sind, zu überbrücken. Die vorrangigen Ziele des Designs sind eine hohe Energieeffizienz, ein geringer Speicherverbrauch, Modularität, sowie eine einheitliche, von der zugrundeliegenden Hardware unabhängigen, API Schnittstelle.

Der Kernel basiert auf dem FireKernel [21], implementiert eine Mikrokernel-Architektur und ist demzufolge modular. Durch die Modularisierung kann das System auf einen spezifischen Einsatz zugeschnitten werden, wodurch der Speicherverbrauch minimiert wird. Abhängigkeitn zwischen den einzelnen Modulen werden weitestgehend vermieden. Bei dem Scheduler handelt es sich um einen *tickless scheduler*, welcher in den Leerlauf-Modus (*idle thread*) übergeht sobald keine Tasks mehr zur weiteren Verarbeitung zur Verfügung stehen. Dieser idle thread kann nur durch Unterbrechnungssignale verlassen werden. Dadurch, dass die maximale Zeit im idle thread verbracht wird, wird der Stromverbrauch des gesamten Systems minimiert. Durch die Unterstützung der Programmiersprachen C sowie C++, wodurch externe Bibliotheken problemlos eingebunden werden können, sowie der GNU Compiler Collection (GCC) wird eine hohe Benutzerfreundlichkeit für Entwickler erreicht. POSIX Konformität ist ebenfalls (teils) gegeben. RIOT kann für *NIX Systeme kompiliert werden, wodurch die Entwicklung zum Großteil in einer Linux Umgebung stattfinden kann, bevor die Software auf die letztendliche Hardware gespielt wird. Trotz des großen Umfangs an Funktionen, hat RIOT einen sehr kleinen Speicherbedarf. Eine einfache Anwendung benötigt lediglich 5 Kilobyte Festwertspeicher sowie 2 Kilobyte Arbeitsspeicher [1].

In Tabelle 2.1 werden die oben genannten Eigenschaften zusammengefasst und gegenübergestellt.

Tabelle 2.1: Schlüsselaspekte von Contiki, Tiny OS, Linux und Riot

OS	Min RAM	Min ROM	C Support	C++ Support	Multi-Threading	MCU w/o MMU	Modularity	Real-Time
Contiki	<2kB	<30kB	○	✗	○	✓	○	○
Tiny OS	<1kB	<4kB	✗	✗	○	✓	✗	✗
Linux	∼1MB	∼1MB	✓	✓	✓	✗	○	○
RIOT	∼1.5kB	∼5kB	✓	✓	✓	✓	✓	✓

(✓) unterstützt, (○) teilweise unterstützt, (✗) nicht unterstützt

RIOT enthält zusätzlich einen vollständigen Network Stack, den Generic (GNRC) Network Stack. Bei dem streng modular aufgebautem GNRC Network Stack wird jedem Modul, demzufolge auch jedem Protokoll, ein eigener Thread zugewiesen. Die Kommunikation unterhalb dieser Threads erfolgt über RIOTs *Inter Process Communication* Modul. Die Struktur dieser Nachrichten ist durch die *netapi* klar definiert. Die einzelnen Module implementieren keine eigenen Warteschlange für Nachrichten (*message queue*, das Inter Process Communication (IPC) Modul allerdings schon. Diese *message queue* ist für die Verwendung des GNRC Network Stack unabdingbar. Einzelnen Threads können über einen entsprechenden Eintrag im Netzwerk Register (*netreg*) Nachrichten eines bestimmten Typs abonnieren [11].

Während eine Nachricht den Network Stack durchläuft, wird sie im Paket Pufferspeicher *pktbuf* hinterlegt.

2.3 Gruppenschlüsselmanagementprotokolle

——— Der folgende Text ist ein direktes Zitat von [22]. ———

n diesem Kapitel werden verschiedene Gruppenschlüsselprotokolle vorgestellt. Letztendlich verfolgen die in diesem Abschnitt beschriebenen Verfahren GKMP, GDOI und G-IKEv2 das selbe Ziel, Gruppenschlüsselmaterialien gesichert an die Gruppenteilnehmer zukommen zu lassen.

——— Ende des Zitats. ———

2.3.1 Group Key Management Protocol

——— Der folgende Text ist ein direktes Zitat von [22]. ———

as Group Key Management Protocol (GKMP) ist ein wohlbekanntes Verfahren, das im RFC 2093 und 2094 beschrieben wird. Im GKMP existieren zwei Rollen: der sogenannte Group Controller (GC) und der Gruppenmitglied (GM). Der Group Controller ist ein Gruppenmitglied mit höherer Autorität und übernimmt die Verwaltung kritischer Gruppenaktionen. Er erstellt den Gruppenschlüssel, verteilt diesen an die Gruppenmitglieder und übernimmt ebenso die Verwaltung des Rekey Prozesses. Mit jedem Gruppenmitglied teilt sich der Group Controller jeweils einen geheimen Schlüssel, den Key Encryption Key (KEK).

Dadurch, dass beim GKMP mit jedem Mitglied ein persönlicher Schlüssel ausgehandelt wird, kann bei der Erzeugung einer Gruppe der Group Key Paket (GKP) verschlüsselt mit dem KEK jedes Teilnehmers per Unicast an ihnen verschickt werden. Der GKP besteht aus einem GTEK und einem GKEK. Der GTEK ist dafür da, um Gruppennachrichten zu ver- und entschlüsseln. Der GKEK ist dafür da, um den GTEK zu ver- und entschlüsseln.

Tritt ein neues Mitglied in die Gruppe ein, kann der GKP verschlüsselt mit seinem privaten KEK übertragen werden. Die übrigen Clients erhalten das GKP verschlüsselt mit dem bisherigen GKEK. So ist die Backward Secrecy sichergestellt.

Ist ein Rekey notwendig, wird ein neues GKP erstellt und mit dem bisherigen GKEK verschlüsselt und mittels einer Broacast Nachricht an die Gruppenmitglieder veteilt.

Tritt ein Teilnehmer aus einer Gruppe aus, so wird ein neues GKP bestehend aus einem neuen GKEK und einem neuen GKEK erstellt, um Forward Secrecy zu gewährleisten. Der neue GKP wird anschließend jeweils mit dem KEK der Teilnehmer, die noch in der Gruppe sind, verschlüsselt und per Unicast an ihnen versendet. Der Austritt beinhaltet einen großen Nachrichtenaufwand, was im Netzkapazität des MANET unvorteilhaft ist.

In dieser Arbeit wird die Beitrittsoperation wie oben beschrieben durchgeführt. So sind aus Sicht des zentralen Servers zwei Nachrichten an die Gruppenteilnehmer zu verteilen, um Backward Secrecy zu erreichen.

——— Ende des Zitats. ———

2.3.2 Group Domain of Interpretation

GDOI ist ein Protokoll zur Verwaltung von Gruppenschlüsseln, bei dem ein Group Controller / Key Server (GCKS) Sicherheitszuweisungen, also kryptographische Richtlinien und Schlüsselmaterial, verteilt. Das Protokoll basiert auf Internet Security Association and Key Management Prokoll (ISAKMP) [14] und IKE Version 1 [7] und wird in RFC 6407 [6] spezifiziert. Die in RFC 4046 („Multicast Security (MSEC) Group Key Management Architecture“ [2]) festgelegten Voraussetzungen werden von GDOI erfüllt. ISAKMP definiert zwei Verhandlungsphasen zwischen zwei Einheiten. In der ersten Phase einigen sich beide Einheiten darauf, wie die weitere Kommunikation untereinander gesichert werden soll; hierbei entsteht eine ISAKMP Security Association (SA). In der zweiten Verhandlungsphase werden weitere SAs generiert, die zur Sicherung von weiteren Protokollen genutzt werden können.

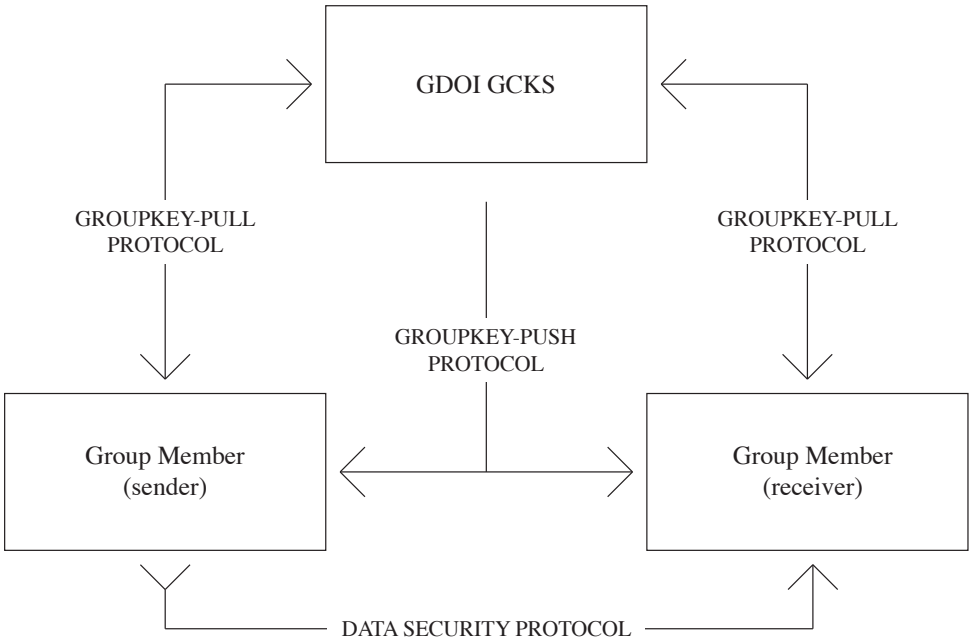

Abbildung 2.1: GDOI Management Model

GDOI erweitert ISAKMP um zwei Protokolle in Phase 2 (Abbildung 2.1):

GROUPKEY-PULL Bei diesem Nachrichtenaustausch wird der Abruf der SAs vom Gruppenmitglied (GM) initiiert. Dieser ist wie oben beschrieben durch ein ISAKMP Phase 1 Protokoll geschützt. Nach dem Austasch ist das Gruppenmitglied, durch den Erhalt der SAs, in der Lage an einer gesichterten Gruppenkommunikation teilzuhaben.

GROUPKEY-PUSH Beim GROUPKEY-PUSH hingegen initiiert der GCKS den Nachrichtenaustausch. Dieser findet in der Regel über Multicast statt und kann somit alle Gruppenmitglieder erreichen. Durch den Austausch können hierbei Mitglieder von der

weiteren Gruppenkommunikation ausgeschlossen werden, in dem ihnen durch die versendeten Gruppenrichtlinien die entsprechende Autorisierung entzogen wird.

2.3.3 Group Internet Key Exchange version 2

——— Der folgende Text ist ein direktes Zitat von [22]. ———

er G-IKEv2 ist ein Gruppenschlüsselmanagementprotokoll, das sich auf Internet Key Exchange version 2 (IKEv2) stützt [20]. IKEv2 ist ein IPSec basiertes Protokoll und schafft eine sichere Punkt zu Punkt Übertragung zwischen zwei Kommunikationspartnern.

G-IKEv2 definiert ein ähnliches Gruppenschlüsselprotokoll wie GDOI, wobei dieser IKEv1 verwendet. Ebenso wie GDOI richtet sich G-IKEv2 an den im RFC3740 beschrieben "Multicast Group (MEC) Security Architecture" und den "Multicast Security (MSEC) Group Key Management Architecture" im RFC4046.

G-IKEv2 behält die initialen Nachrichtenaustausche des IKEv2 bei und besteht somit initial aus einer IKE_SA_INIT Phase, um einen sicheren Kanal zwischen dem Client und dem vertrauenswürdigen GCKS zu erstellen. Der Initiator fragt am GCKS mit einer Security Association, einem Key Exchange und einer sogenannten Nonce (eine zufällige Zahl) an. Die Security Association umfasst unter anderem, welche Verschlüsselungsalgorithmen verwendet werden sollen. Der Key Exchange beinhaltet den Diffie Hellman Wert des Initiators. Die Antwort vom GCKS beinhaltet dieselben Datenstrukturen mit anderen Werten. Nach dieser Initialisierungsphase werden von beiden Parteien der gemeinsame Schlüssel berechnet, mithilfe dessen die darauffolgenden Nachrichten verschlüsselt übertragen werden.

Anschließend folgt die GSA_AUTH Phase wieder mit zwei Nachrichten, in der sich der Client authentifizieren kann. Hier kann entweder mit einem Pre Shared Key (PSK) oder einem Zertifikat vorgegangen werden, um die Authentifizierung festzustellen. Diese vier Nachrichtenaustausche illustrieren quasi die Anmeldung eines Clients beim Server. Anschließend folgen mit dem GSA_REKEY-Austausch Rekey Informationen. Diese werden per Multicast nur vom GCKS zu den Clients übertragen und haben keine Erwiderung vom Client zum GCKS.

Der Unterschied von IKEv2 zu IKEv1 liegt unter anderem darin, dass IKEv2 in einem einzigem RFC [RFC7296] spezifiziert wird und die RFCs 2407, 2408 und 2409, in denen IKEv1 beschrieben wird, ersetzt. Vor allem aber wurden die initialen Nachrichtenaustausche von acht auf vier reduziert [10], was im Bereich des IdD große Vorteile mit sich bringt.

Einleitend mit einem sicheren Kommunikationskanal, das sich am IKEv2 anlehnt, wurde zusammen mit anderen Verfahren ein hybrides Gruppenschlüsselmanagementprotokoll konzipiert, das neben effizienterem Rekey wenig Nachrichtenaufwand beinhaltet und dabei hohen Wert auf Forward und Backward Secrecy bei den einzelnen Gruppenoperationen gelegt wird.

——— Ende des Zitats. ———

2.4 Gruppenschlüsselmanagement

In diesem Kapitel werden zwei Schlüsselmanagementverfahren erläutert. Kapitel 2.4.1 befasst sich mit dem LKH, Kapitel 2.4.2 beschreibt das SL.

LKH ermöglicht einen geringeren Rechenaufwand bei der Schlüsselerzeung, da hierbei meist nur Teilbäume des Schlüsselbaums berücksichtigt werden müssen. Dank dem Secure Lock ist es möglich, den Gruppenschlüssel mit einer einzigen Broadcast Nachricht an die entsprechenden Teilnehmer einer Gruppe zu verteilen. Dabei können unauthorisierte Teilnehmer, die das Datenpaket erhalten, das Secure Lock nicht entschlüsseln, da ihre persönlichen Schlüssel nicht in die Berechnungen des CRT eingeflossen sind.

CAKE vereint Eigenschaften aus beiden Protokollen, um die Berechnungen des CRT effizienter zu gestalten und den Nachrichtenaufwand bei Gruppenoperationen zu minimieren.

2.4.1 Logical Key Hierarchy

——— Der folgende Text ist ein direktes Zitat von [22]. ———

as LKH basiert auf einen Schlüsselbaum (siehe Abbildung 2.2), der vom GCKS verwaltet wird. Jeder Knoten beinhaltet einen KEK. Die Blattknoten repräsentieren die einzelnen Gruppenmitglieder und jedes Blattknoten beinhaltet einen KEK, den entsprechenden persönlichen Schlüssel des jeweiligen Teilnehmers. Jedem Gruppenmitglied wird neben seinem privaten Schlüssel zudem die KEKs aller Knoten, die auf seinem Pfad von seinem Knoten bis zur Wurzel liegen, mitgeteilt.

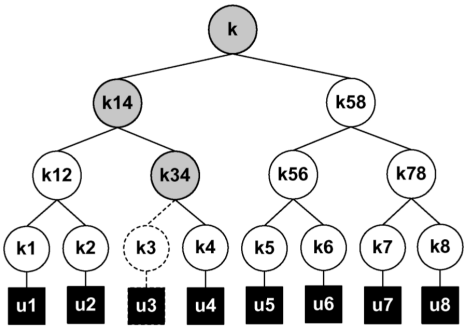

Abbildung 2.2: Der LKH Schlüsselbaum aus [16]

Betritt ein neuer Teilnehmer u3 die Gruppe, so wird ein neuer Blattknoten im Schlüsselbaum hinzugefügt (in Abbildung 2.2 gepunktet dargestellt). Damit der neue Teilnehmer nicht in Besitz des bisherigen Gruppenschlüssels kommt und somit die Backward Secrecy aufrecht zu erhalten, müssen die betroffenen Knoten k, k14, k34, die auf dem Pfad des neuen Teilnehmers liegen, aktualisiert werden. Es werden neue Schlüssel k', k'14 und k'34 generiert und anschließend wird von der Wurzel aus die Schlüssel mit den jeweiligen Kindknoten verschlüsselt: k wird mit k'14 und k58 verschlüsselt, k'14 wird mit k12 und k'34 verschlüsselt, k'34 wird mit k3 und k4 verschlüsselt. Anschließend werden die Gruppenteilnehmer über die neuen entsprechenden KEKs informiert.
Der Austritt aus der Gruppe funktioniert ähnlich. Wenn ein Teilnehmer u3 austritt, werden die KEKs aus den Knoten, die auf seinem Pfad bis zur Wurzel liegen, aktualisiert. Im Anschluss werden die neuen Schlüssel k', k'14 und k'34 jeweils mit den entsprechenden Schlüssel aus den Kindknoten verschlüsselt. k'34 wird jedoch nur mit k4 verschlüsselt. Somit kann u3 nicht mehr k34 entschlüsseln und Forward Secrecy ist sichergestellt.

——— Ende des Zitats. ———

2.4.2 Secure Lock

Bei dem Secure Lock Verfahren wird zunächst eine Nachricht M mit einem Session Key d verschlüsselt. Der Session Key wird hierbei bei jeder zu versendenen Nachricht neu generiert. Das Secure Lock x wird nun dazu verwendet, den *session key* zu verschlüsseln. Die verschlüsselte Nachricht wird im Anschluss an den verschlüsselten Session Key angehängt, woraus sich der folgende in Abbildung 2.3 dargestellte Aufbau ergibt.

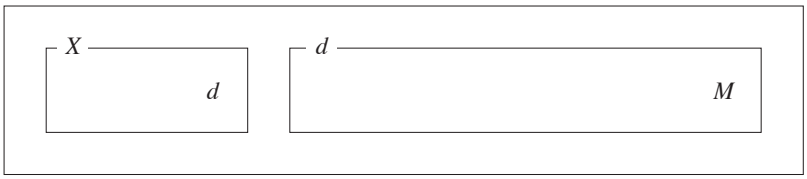

Abbildung 2.3: Aufbau einer mit dem Secure Lock gesicherten Nachricht

Das Secure Lock x muss dabei die folgenden Eigenschaften erfüllen:

- Es darf nur von denjenigen Nutzern geöffnet werden können, für die die Nachricht bestimmt ist
- Es muss vom Session Key d abhängig sein

Um beide dieser Eigenschaften zu erfüllen, muss eine funktionelle Abhängigkeit zwischen dem Secure Lock und dem eigentlichen Schlüssel, als auch zwischen dem Secure Lock und dem Session key bestehen. Im weiteren Verlauf dieses Kapitels wird erläutert, wie ein solches Secure Lock unter Verwendung des Chinesichen Restsatzes (im Folgenden *CRT*) konstruiert werden kann.

Seien $N_1, N_2, \ldots, N_n$ natürliche, paarweise teilerfremde Zahlen, $R_1, R_2, \ldots, R_n$ natürliche Zahlen und $L = N_1 * N_2 * \ldots * N_n$, dann gibt es ein System von Kongruenzen

$$\begin{aligned} X &\equiv R_1 \; mod \, N_1 \\ X &\equiv R_i \; mod \, N_i \\ X &\equiv R_n \; mod \, N_n \end{aligned}$$

mit einer gemeinsamen Lösung X im Intervall $[1, L-1]$ und

$$\begin{aligned} x &= \left(\sum_{i=1}^{n} (L/N_i) * R_i * f_i \right) \; mod \, L \\ \text{mit} \quad & 1 \equiv f_i * (L/N_i) \; mod \, N_i), \; R_i \leq N_i. \end{aligned}$$

Beim Secure Lock ist nun $R_i = E_{ek_i}(d)$, wobei E die Verschlüsselungsoperation und ek_i den privaten Schlüssel eines Nutzers u_i notiert.

$$\begin{aligned} X &\equiv (E_{ek_1}(d)) \; mod \, N_1 \\ X &\equiv (E_{ek_i}(d)) \; mod \, N_i \\ X &\equiv (E_{ek_n}(d)) \; mod \, N_n \end{aligned}$$

Ein Nutzer u_i, dessen privater Schlüssel ek_i und natürliche Zahl N_i mit in die Berechnung eingeflossen ist, kann über die Umkehrfunktion $X \; mod \; N_i$ seinen Wert R_i berechnen, aus welchem er wiederum den Session Key d ableiten, und letztendlich die Nachricht M entschlüsseln kann [3].

3 Central Authorized Key Extension - CAKE

——— Der folgende Text ist ein direktes Zitat von [22]. ———

CAKE ist ein hybrides Gruppenschlüsselmanagementverfahren, das für eine sichere Kommunikation innerhalb einer Gruppe sorgt, wobei strikte Sicherheitsanforderungen und minimale Netzbelastung im Vordergrund stehen. Central Authorized Key Extension (CAKE) erfüllt diese Voraussetzungen mit minimalem Kommunikations- und Berechnungsaufwand. Die Schlüsselverwaltung ist zentral vom CAKE GCKS organisiert. Der CAKE GCKS führt die Schlüsselberechnungen durch und verteilt die Schlüssel an die entsprechenden Mitglieder einer spezifizierten Gruppe. Im Abschnitt 3.1 folgen weitere Erklärungen um das CAKE System und im Abschnitt 3.2 werden Anforderungen aufgezählt, die CAKE benötigt.

——— Ende des Zitats. ———

3.1 Definition und Erklärung

——— Der folgende Text ist ein direktes Zitat von [22]. ———

ur authorisierte Teilnehmer können im CAKE System einer gesicherten Gruppenkommunikation beitreten. Dazu müssen sie sich dementsprechend authentifizieren. In dieser Arbeit geschieht dies durch einen PSK. Hinzu kommt, dass jeder neuen Teilnehmer, der die Registrierungspahse durchläuft, zwei geheime Schlüssel zugewiesen bekommt, einen KEK und eine Primzahl. Mit dem persönlichen KEK werden Nachrichten vom GCKS zum Client abgesichert. Zudem fließen beide Schlüssel in die Berechnungen des CRT mit ein.
Im CAKE System existiert außerdem das Gruppenschlüsselpaar aus GTEK und GKEK. Der GTEK wird zur Verschlüsselung von Gruppennachrichten benutzt und mit dem GKEK wird der GTEK abgesichert.

——— Ende des Zitats. ———

Unterteilung in GTEK und GKEK

——— Der folgende Text ist ein direktes Zitat von [22]. ———

in separater GKEK zur Verbreitung eines neuen GTEK ist notwendig, da die Nutzung des aktuellen $\text{GTEK}_{\text{aktuell}}$ als $\text{GKEK}_{\text{aktuell}}$ bei bitweisen XOR dazu führt, dass der neue Teilnehmer die alten Nachrichten entschlüsseln könnte. Durch Anwendung von XOR auf die abgefangene Nachricht mit GTEK_{neu} ergibt sich $\text{GTEK}_{\text{aktuell}}$. Dies wird durch den Einsatz eines separaten GKEK verhindert.

——— Ende des Zitats. ———

3.2 Anforderungen von CAKE

Das in dieser Arbeit präsentierte Protokoll stellt verschiedene Anforderungen an das zugrunde liegende System. Diese Anforderungen können in Sicherheitsanforderungen, funktionale und nicht-funktionale Anforderungen unterteilt werden.

3.2.1 Sicherheitsanforderungen

——— Der folgende Text ist ein direktes Zitat von [22]. ———

u den sicherzustellenden Schutzzielen zählen Forward und Backward Secrecy, Schlüsselunabhängigkeit, Kollisionsfreiheit. Zur Informationssicherheit gehören zudem Authentizität, Vertraulichkeit und Integrität [19].

——— Ende des Zitats. ———

Forward Secrecy

——— Der folgende Text ist ein direktes Zitat von [22]. ———

ritt ein Teilnehmer in eine bestehende Gruppe ein, so darf er nicht in der Lage sein, die im Vorhinein empfangenen Nachrichten zu entschlüssen.

——— Ende des Zitats. ———

Backward Secrecy

——— Der folgende Text ist ein direktes Zitat von [22]. ———

erlässt ein Teilnehmer eine gesicherte Gruppenkommunikation, so darf er nicht in der Lage sein, im Nachhinein empfangenen Nachrichten zu entschlüsseln.

——— Ende des Zitats. ———

Schlüsselunabhängigkeit / Folgenlosigkeit

——— Der folgende Text ist ein direktes Zitat von [22]. ———

urch den Besitz eines Schlüssels darf es nicht möglich sein, auf andere Schlüssel zu schließen.

——— Ende des Zitats. ———

Kollisionsfreiheit

——— Der folgende Text ist ein direktes Zitat von [22]. ———

egeht eine Teilmenge einer Gruppe ein unerlaubtes Vorgehen, so führt das nicht zum Schaden eines einzelnen Gruppenmitglieds.

——— Ende des Zitats. ———

Authentizität

Das Protokoll muss Verfahren umfassen, die es sowohl den Clients als auch der Server erlauben, sich gegenseitig Vertrauenswürdigkeit und Echtheit ihrer Nachrichten zu beweisen.

Vertraulichkeit auf Gruppenebene

——— Der folgende Text ist ein direktes Zitat von [22]. ———

ruppenschlüsselinformationen, die für eine Gruppe spezifiziert sind und weitere Daten, die innerhalb der Gruppe ausgestauscht werden, dürfen von Externen nicht gelesen werden können.

——— Ende des Zitats. ———

Datenintegrität

——— Der folgende Text ist ein direktes Zitat von [22]. ———

icherstellung, dass Daten beim Übertragen nicht verändert werden.

——— Ende des Zitats. ———

3.2.2 Funktionale Anforderungen

Funktionale Anforderungen beschreiben die Funktionaliäten, die eine Implementierung zur Verfügung stellen muss. Das System um CAKE muss in der Lage sein, Nachrichten zu versenden und zu empfangen. Es muss Funktionen zur Erstellung, Löschung und Verwaltung von Gruppen ermöglichen. Durch die Verwendung von RIOT und dem dazugehörigen GNRC Network Stack ist diese Anforderung erfüllt. Diese Nachrichten müssen außerdem mittels kryptographischer Verfahren verschlüsselt und entschlüsselt werden können. Ebenfalls muss es in der Lage sein das CRT zu berechnen und aufzulösen, sowie geeignetes Schlüsselmaterial zu erzeugen. Im Folgenden werden weitere Anforderungen prägnant aufgezählt, die ein Gruppenschlüsselprotokoll benötigt.

Multicast Kommunikation

——— Der folgende Text ist ein direktes Zitat von [22]. ———

er GCKS sollte neben Unicast Nachrichten ebenso broadcastbasierte übertragen können, um Gruppenfunktionen zu verwalten und Schlüsselinformationen zu verschicken.

——— Ende des Zitats. ———

Gruppenoperationen

——— Der folgende Text ist ein direktes Zitat von [22]. ———

in modernes Gruppenschlüsselmanagementprotokoll deckt folgende Gruppenoperationen ab, wobei die Sicherheitsanforderungen nicht missachtet werden dürfen. Zunächst werden die einzelnen Operationen vorgestellt. Im weiteren Verlauf dieses Kapitels werden sie näher erläutert.

- **Einzel-Eintritt**
  Ein Teilnehmer tritt in eine bestehende Gruppe ein. Dabei muss die Backward Secrecy gewährleistet sein.
- **Mehrfach-Eintritt**
  Mehrere Teilnehmer treten in eine bestehende Gruppe ein. Dabei muss die Backward Secrecy gewährleistet sein.
- **Einzel-Austritt / Einzel-Rausschmiss**
  Ein Gruppenmitglied tritt aus der Gruppe aus oder ein suspekter Gruppenmitglied wird vom Gruppenmanager aus einer Gruppe entfernt. Dabei muss die Forward Secrecy beachtet werden.
- **Mehrfach-Austritt / Mehrfach-Rausschmiss**
  Mehrere Gruppenmitglieder treten aus der Gruppe aus bzw. werden aus dem CAKE System rausgeschmissen. Dabei muss die Forward Secrecy beachtet werden.
- **Re-Keying**
  Die Aktualisierung des Gruppenschlüssels muss über eine effiziente Vorgehensweise möglich sein. Gleichzeitig darf die Forward und Backward Secrecy nicht gefährdet sein.
- **Gruppenverschmelzung**
  Mehreren Gruppen ist durch Re-Keying ein gemeinsamer Schlüssel effizient bereitzustellen. Dabei muss die Backward Secrecy gewährleistet sein.
- **Gruppenteilung**
  Eine Gruppe teilt sich in mehrere Teilgruppen auf. Die Forward Secrecy muss hier ebenfalls sichergestellt sein.

—— Ende des Zitats. ——

Kryptographische Anforderungen

Die Verschlüsselung von Nachrichten ist für das Protokoll unabdingbar, da es vor allem für Funktechnologien konzipiert wurde, wobei Nachrichten leicht abgefangen werden können. Dies setzt bestimmte Bedingungen voraus. Diese beinhalten die Benutzung von mindestens einem Verschlüsselungsalgorithmus, einem Algorithmus für Integritätswahrung, einen Generator für Zufallszahlen ausreichender Güte und eine Diffie-Hellman Gruppe für den Aufbau eines sicheren Kanals zwischen Client und Server.

3.2.3 Nicht-funktionale Anforderungen

Das Protokoll muss einer Reihe von nicht-funktionalen Anforderungen genügen. Diese sind in Rechenleistung, Speicherkapazität und Eneergieffizienz gegliedert.

Zuverlässigkeit

Das System muss eventuell auftretende Fehler abfangen, verarbeiten und in der Lage sein, sich entsprechend wieder zu erholen.

Skalierbarkeit

Das Protokoll muss in der Lage sein, eine größere Anzahl an Clients pro Gruppe zu unterstützen. Die zu unterstützende Anzahl wird auf 81 Teilnehmer begrenzt.

Rechenleistung

Da die eingesetzten Geräte über eine geringe Rechenleistung verfügen, muss der benötigte Aufwand für die Verschlüsselung, Entschlüsselung sowie die Erzeugung von Schlüsselmaterial, insbesondere durch das CRT, möglichst gering gehalten werden.

Speicherkapazität

Der verfügbare Arbeitsspeicher auf den Geräten muss ausreichend groß sein, um das Betriebssystem RIOT laden zu können. Hinzu kommt der für das Schlüsselmaterial, die Gruppen, und die Clients benötigte Speicher. Der Client sollte über ausreichend Arbeitsspeicher verfügen, um mindestens einer Gruppe beitreten zu können.

Energieeffizient

Das Protokoll ist für kompakte, portable Geräte ohne konstante Stromversorgung konzipiert. Als solches muss die Anzahl und der Aufwand der benötigten Rechenoperationen, damit auch der Stromverbrauch, möglichst gering gehalten werden.

4 Design

Dieses Kapitel dient der Beschreibung der Architektur und des Designs des Protokolls, in das CAKE eingegliedert wurde, unter Einbezug der in Kapitel 3.2 definierten Anforderungen.

Das Protokoll benutzt den UDP Port 849, da dieser zum Entstehungszeitpunkt dieser Arbeit noch nicht von der *Service Name and Transport Protocol Port Number Registry* vergeben wurde und somit zur freien Verfügung steht. Das G-IKEv2 Protokoll [8], an das sich dieses Protokoll teilweise anlehnt, benutzt UDP Port 848.

Nach der initialen Registrierung eines Clients, ist dieser in der Lage die Erstellung einer neuen Gruppe anzufordern, Gruppen beizutreten sowie Nachrichten zu verschicken. Die Registrierung, sowie die für die Gruppenverwaltung notwendigen Nachrichten werden via Unicast übermittelt. Der Client erwartet auf jede von ihm an den Server übermittelten Unicast-Nachricht eine Antwort. Diese enthält, im Falle einer erfolgreich ausgeführt Aktion, die angeforderten Daten, oder, im Falle eines Verarbeitungsfehlers einen entsprechenden Fehler-Code. Nachrichten innerhalb einer Gruppe, sowie einige der für die Schlüsselverteilung notwendigen Nachrichten werden mittels Multicast übermittelt und müssen nicht quittiert werden.

Kapitel 4.1 beschreibt die verschiedenen Rollen innerhalb des Protokolls, Kapitel 4.2 beschreibt den Aufbau eines sicheren Kanals zwischen einem Client und dem GCKS, in Kapitel 4.3 werden die verschiedenen Gruppenoperationen erläutert.

4.1 Rollenmodell

Aus dem in Kapitel 2 beschriebenen Szenario und den definierten Anforderungen lassen sich vier verschiedene Rollen ableiten: (i) Key Server, (ii) Group Controller, (iii) Client und (iv) Group Manager.

(i) Key Server Die wesentliche Aufgabe des Servers besteht darin, das Schlüsselmaterial sowie die Schlüsselbäume zu verwalten. Er ist für die Berechnung und Erneuerung des Schlüsselmaterials verantworlich.

(ii) Group Controller Der Group Controller hat die Aufgabe alle Clients und Gruppen zu verwalten. Bei Bedarf fordert er den Key Server dazu auf, neues Schlüsselmaterial zu berechnen, welches er an die Clients verteilt.

(iii) Client Der Client entspricht dem tatsächlichen Gruppenmitglied. Als solcher kann er Schlüsselmaterial empfangen, sowie Nachrichten empfangen und versenden.

(iv) Group Manager Für die organisatorischen Bedürfnisse innerhalb einer Gruppe ist der Group Manager zuständig. Abhängig von der Implementierung ist dieser nicht zwingend selber ein Mitglied der Gruppe, die er verwaltet. Der Group Manager kann,

zusätzlich zu den Berechtigungen eines normalen Clients, Clients aus der Gruppe ausschließen, die Löschung seiner Gruppe beantragen sowie den Key Server dazu anstoßen, einen Rekey-Prozess einzuleiten.

Der Nachrichtenaustausch von der Registrierung eines Clients bis hin zur ersten Gruppenoperation wird in Abbildung 4.1 veranschaulicht.

4.2 Aufbau eines sicheren Kanals

Der Aufbau des sicheren Kanals zwischen dem GCKS und einem Client ist stark an das G-IKEv2-Protokoll angelehnt und besteht aus einem CAKE_INIT sowie einem CAKE_AUTH Nachrichtenaustausch. Die einzelnen Payloads dieser Nachrichten werden in Kapitel 4.2.1 und 4.2.2 beschrieben und in Abbildung 4.2 dargestellt.

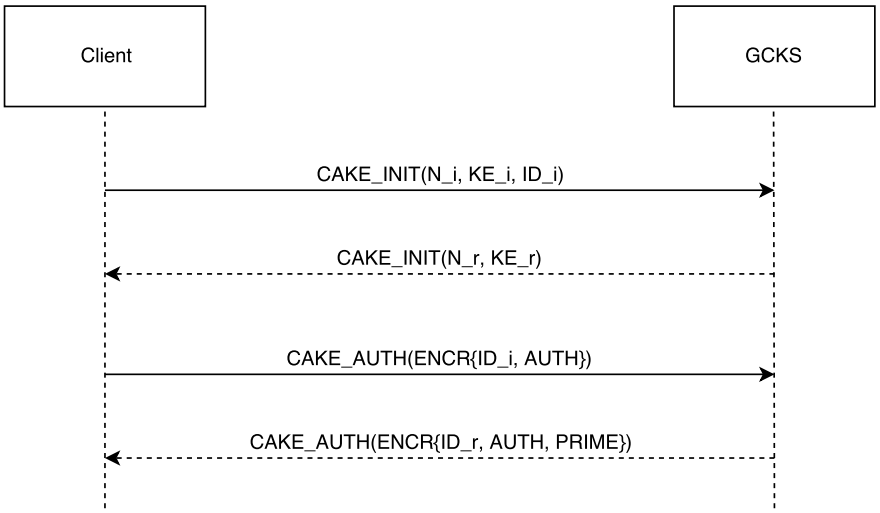

Abbildung 4.2: Aufbau eines sicheren Kanals

4.2.1 CAKE_INIT

Der ersten Nachrichten des Protokolls sind die zwei CAKE_INIT Nachrichten, die sich der registrierende Client und der GCKS zuschicken. Der Nachrichtenaustausch wird hierbei stets vom Client initiiert. Der Client schickt dem GCKS die folgenden Payloads: eine Zufallszahl (Nonce) N_i und den Wert für den Diffie-Hellman-Schlüsselaustausch KE_i. Die kryptographischen Algorithmen werden derzeit nicht zwischen Client und GCKS ausgehandelt, sondern in der entsprechenden Konfigurations-Datei definiert. Der GCKS antwortet auf diese Nachricht mit seiner Nonce N_r und seinem Diffie-Hellman-Wert KE_r. Beide Parteien sind nun in der Lage, ein gemeinsames Geheimnis zu berechnen, aus welchem letztendlich das Schlüsselmaterial für die Verschlüsselung der folgenden Kommunikation abgeleitet wird.

4.2.2 CAKE_AUTH

Um sich zu authentifizieren sendet der Client den Identitäts-Payload ID_i sowie einen Authentifizierungs-Payload $AUTH$, der das für die Authentifizierung benötigte Geheimnis,

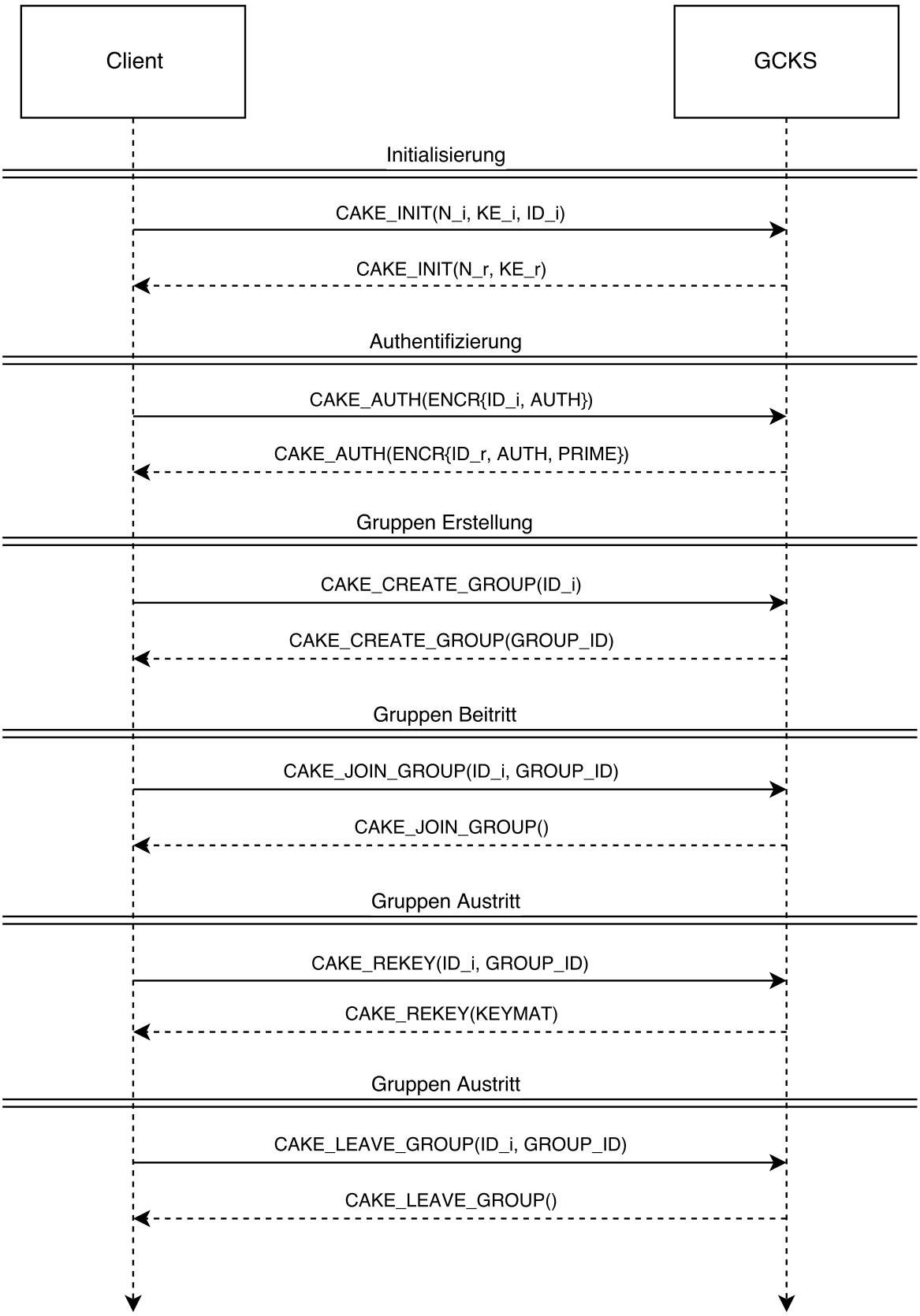

Abbildung 4.1: Sequenzdiagram

den Pre Shared Key (PSK), enthält. Diese Payloads werden mit dem zuvor abgeleiteten Schlüsselmaterial gesichert. Der PSK wird vom GCKS überprüft, welcher bei einer Übereinstimmung seinerseits dem Client den Identitäts-Payload ID_r, den Authentifizierungs-Payload $AUTH$ sowie eine Primzahl schickt. Hierbei wird für jede Kommunikationsrichtung ein eigener Schlüssel verwendet.

4.3 Gruppenoperationen

——— Der folgende Text ist ein direktes Zitat von [22]. ———

m beschriebenen Szenario ist es von großer Bedeutung, eine Gruppe sicher und rasch zu erstellen. Hinzu kommt, dass Teilnehmer sowohl relativ unkompliziert in eine Gruppenkommunikation hinzugefügt werden sollen, als auch welche zu entfernen. Diese zwei Operationen sind grundlegend wichtig für ein Gruppenschlüsselprotokoll und dürfen in diesem Kontext nicht fehlen. In diesem Kapitel werden diese Aspekte des CAKE Protokolls ausführlich erklärt. Im Abschnitt 4.3.1 wird beschrieben, wie der GCKS im CAKE System eine Gruppe erzeugt. Der Einzeleintritt und der Einzelaustritt werden im Abschnitt 4.3.2 bzw. 4.3.3 erläutert. In 4.3.4 werden die Operationen Mehrfacheintritt und Mehrfachaustritt fomuliert und die Aktualisierung des Gruppenschlüssels wird unter "'Rekey einer Gruppe"' 4.3.5 beschrieben.

——— Ende des Zitats. ———

4.3.1 Initiale Erzeugung einer Gruppe

——— Der folgende Text ist ein direktes Zitat von [22]. ———

n CAKE sind die grundlegenden Informationen einer Gruppe der GTEK und der GKEK. Dieses Schlüsselpaar muss gesichert an die entsprechenden Teilnehmer einer spezifizierten Gruppe verteilt werden, die vorher bereits im CAKE System registriert sein müssen.

Dazu erstellt der Key Server zufällig einen GTEK und einen GKEK. Für die initiale Schlüsselverteilung wird ein CRT berechnet. Die Primzahlen aller Teilnehmer fließen in die Berechnungen des CRT ein, indem sie zunächst miteinander multipliziert werden. Als zweiter Input für den CRT fließt der GKEK verschlüsselt mit dem jeweiligen persönlichen KEK jedes Teilnehmers mit ein. Anschließend werden sie ebenfalls miteinander multipliziert. Entscheidend dabei für den weiteren Rechenvorgang des CRT ist, dass bis hierhin die Primzahlen der Teilnehmer in der selben Reihenfolge wie der jeweils mit dem KEK jedes Teilnehmers veschlüsselte GKEK bereitstehen. Das bedeutet, die Liste der Primzahlen, die jeweils für einen Client stehen, müssen in der geichen Reihenfolge sein wie die Verschlüsselung des GKEKs mit jedem KEK der Teilnehmer. Es folgen nämlich weitere Multiplikationen und Modulo Berechungen mittels der Primzahl- und GKEK-Listen. Danach ist der Output des CRT, der Secure Lock, abgeschlossen. Nur so kann das broadcastbasierte Datagramm auf Seiten des Clients aufgelöst werden und zum GKEK führen, denn er ist nur im Besitz seines KEKs und seiner Primzahl. Der Secure Lock bildet den ersten Teil des zu versendenen Datagramms. Der GTEK wird mit dem GKEK XOR verschlüsselt und vervollständigt den zweiten Teil der Dateneinheit. Das Paket wird per Broadcast an alle Mitglieder der spezifizierten Gruppe versendet.

Ein Client kann das Secure Lock nur dann auflösen, wenn seine geheime Primzahl in der Berechnung des CRT mit enthalten ist. Nachdem ein Gruppenmitglied die Modulo Funktion mithilfe seiner Primzahl auf das empfangene Secure Lock angewendet und so den entsprechenden verschlüsselten GKEK erhalten hat, kann er den GKEK mit seinem privaten Schlüssel entschlüsseln und den GKEK abspeichern. Mithilfe des GKEK bekommt er nun den GTEK, indem die XOR Funktion auf den überschlüsselten GTEK angewendet wird.

Sind die Teilnehmer im CAKE System angemeldet und wurden sie für eine gesicherte Gruppenkommunikation ausgewählt, dann ist nur eine einzige Broadcast Nachricht notwendig, um allen Teilnehmern einer spezifizierten Gruppe über den GTEK zu informieren. Auf der einen Seite besteht bei der initialen Erzeugung einer Gruppe aufgrund der Berechnungen des CRT zwar einen erhöhten Rechenaufwand für den GCKS, dafür beträgt jedoch die Nachrichtenzahl eins und ist im beschriebenen Szenario perfekt geeignet, da Kommunikation über Funk teuer ist.

——— Ende des Zitats. ———

4.3.2 Eintritt von neuen Teilnehmern in eine Gruppe - Join

——— Der folgende Text ist ein direktes Zitat von [22]. ———

ird ein Client in eine gesicherte Gruppenkommunikation hinzugefügt, muss er zunächst die Anmeldungsphasen von CAKE durchlaufen. Hierzu handelt er mit dem GCKS seinen geheimen Schlüssel, den persönlichen KEK aus. Der private KEK ist nur dem GCKS und dem entsprechenden Client bekannt. Außerdem wird dem neuen Client eine eindeutige Primzahl zugewiesen, der in die Berechnungen des CRT miteinfließt. Nachdem der Client beim CAKE-Server registriert und er beim GCKS eine Betrittssanfrage in eine Gruppe gemacht hat, beginnt der Join Algorithmus.

Dazu erstellt der Key Server ein neues Schlüsselpaar $GTEK_{neu}$ und $GKEK_{neu}$. Mithilfe des gehashten $GKEK_{aktuell}$ werden diese jeweils bitweise mit XOR überschlüsselt. Dem neuen Client versendet der Key Server das Schlüsselpaar $GTEK_{neu}$ und $GKEK_{neu}$ verschlüsselt mit dessen ausgehandeltem privaten KEK. Den restlichen Gruppenteilnehmern wird das neue Gruppenschlüsselpaar verschlüsselt mit dem bisherigen $GKEK_{aktuell}$ per Broadcast verschickt.

Bei einer Join Anfrage in CAKE sind aus Sicht des GCKS nur zwei Nachrichten zu verteilen: Eine Unicast an das neue Gruppenmitglied und eine Broadcast an die bisherigen Mitglieder. Die Clients der spezifizierten Gruppe empfangen jeweils eine einzige Nachricht, worin die neuen Gruppenschlüsseln enthalten sind. Der neue Teilnehmer entschlüsselt die empfangene Unicast Nachricht mit seinem privaten KEK und erhält die Gruppenschlüsseln. Die restlichen Teilnehmer der Gruppe entschlüsseln das empfangene Paket mit dem bisherigen $GKEK_{aktuell}$.

——— Ende des Zitats. ———

4.3.3 Austritt von Teilnehmern aus einer Gruppe - Leave

——— Der folgende Text ist ein direktes Zitat von [22]. ———

in Gruppenmitglied kann auf unterschiedliche Weise aus einer gesicherten Gruppenkommunikation aus dem CAKE System entfernt werden. Er kann selbst eine Anfrage beim

Gruppen-Manager stellen, um eine Gruppe zu verlassen oder er wird vom CAKE GCKS selbst entfernt. Nachdem feststeht, dass ein Teilnehmer aus einer Gruppe ausgeschlossen wird, kann das bestehende Gruppenschlüsselpaar ($GTEK_{aktuell}$, $GKEK_{aktuell}$) nicht mehr verwendet werden, da der austretende Teilnehmer im Besitz dessen ist. Damit also die Backward Secrecy gewährleistet wird, muss das Gruppenschlüsselpaar aus GTEK und GKEK erneuert werden. Der Algorithmus des Leave Requests beruht wie bei der initialen Erzeugung einer Gruppe auf den CRT.
Im Gegensatz des initalen CRT gehen aber bei der Leave Operation nicht die Schlüssel jedes Teilnehmers in die Berechnung des CRT ein, sondern nur eine Teilmenge. Somit vermindern sich die Berechnungskosten für den GCKS des CAKE Systems, die in MANETs so gering wie möglich gehalten werden sollten.
Der GCKS verwaltet eine ternäre Baumstruktur, anhand dessen ein verkleinertes CRT berechnet werden kann und die Gruppenschlüsseln aktualisiert werden. Sind die Gruppenteilnehmer über die entsprechenden Baumknoten informiert und haben sie Kenntnis mit welchem KEK, wird nur eine einzige Broadcast Nachricht seitens des GCKS benötigt, um die Gruppenschlüssel an die verbleibenden Gruppenteilnehmer zukommen zu lassen. Im Folgenden wird der Schlüsselbaum beschrieben.

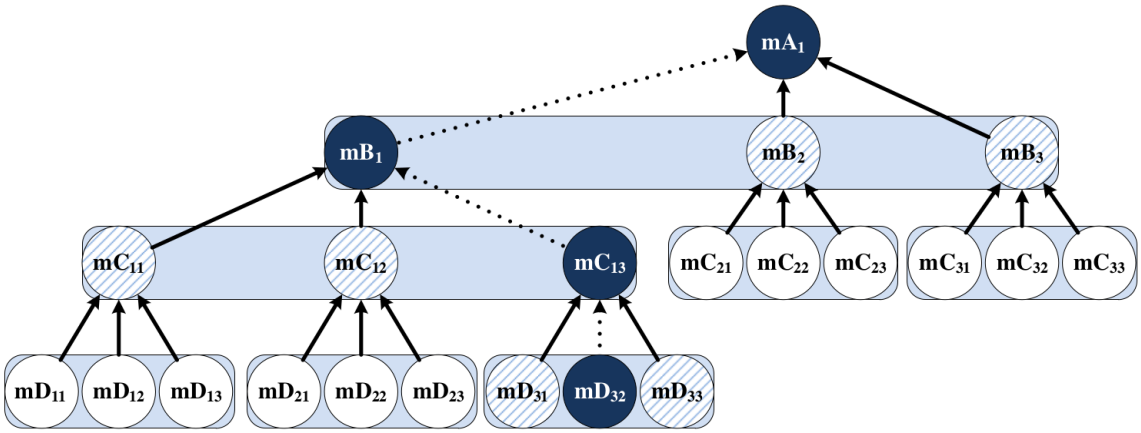

Abbildung 4.3: CAKE Baum

Beim Austritt eines Teilnehmers mD32 wird sein Pfad bis zur Wurzel markiert (in Abbildung 4.3 sind die entsprechenden Knoten dunkel markiert). Die markierten Knoten sind kompromittiert und werden nicht in die folgende CRT Berechnung miteinbezogen. Stattdessen gelangen die im Baum am weitesten oben und nicht markierten Knoten mB2, mB3, mC11, mC12, mD31 und mD33 (in Abbildung 4.3 schraffiert gekennzeichnet) in die Berechnungen des CRT. Dadurch verringert sich in diesem Fall der Rechenaufwand des CRT von 14 Knoten auf sechs. Somit fließt der GKEK sechs mal in den CRT ein. Jedes Mal wird er dabei mit einem der sechs KEKs aus den hingewiesenen Knoten verschlüsselt. Neben der Liste des verschlüsselten GKEK gehen zusätzlich die Primzahlen aus denselben sechs Knoten in den CRT ein. Das Ergebnis des CRT - das Secure Lock - stellt ein Teil des zu versendenden Datagramms dar. Außerdem wird der GTEK bitweise XOR mit dem GKEK überschlüsselt und vervollständigt den zweiten Teil des Pakets.

Da die CRT Berechnung aus dem Schlüsselbaum heraus zeitintensiv und teuer ist, baut der

GCKS immer nur dann die Baumstruktur auf, wenn keine anderen Operationen erforderlich sind. Allein der GCKS hat Kenntnis über die gesamte Baumstruktur und verwaltet sie.

Bei Bedarf hat der GCKS die Möglichkeit, den Baum neu aufbauen und zu balancieren. Dies ist dann der Fall, wenn bereits mehrere Teilnehmer aus einer gesicherten Gruppenkommunikation ausgetreten sind bzw. vom GCKS entfernt wurden. Das bedeutet auch, dass die Baumhöhe sich reduzieren kann. Allerdings sollte dies nur dann geschehen, wenn der GCKS keine anderen Aufgaben bewältigen muss, wie beispielsweise aufgrund von Forward und Backward Secrey die Gruppenschlüssel zu aktualisieren.

Außerdem können die aufgrund von Teilnehmeraustritte entstehenden freien Stellen im Schlüsselbaum mit neuen Blattknoten, die jeweils einen Teilnehmer durch Beitritt in die Gruppe repräsentieren, aufgefüllt werden.

——— Ende des Zitats. ———

4.3.4 Mehrfach Operationen

——— Der folgende Text ist ein direktes Zitat von [22]. ———

n eine Gruppe kann nicht nur ein einzelner Teilnehmer hinzugefügt werden, sondern mehrere Teilnehmer können gleichzeitig mitaufgenommen werden. Ebenso können mehrere Teilnehmer zugleich eine gesicherte Gruppenkommunikation verlassen bzw. entfernt werden. In den folgenden zwei Abschnitten werden diese beiden Aspekte beschrieben.

——— Ende des Zitats. ———

Mehrfach-Eintritt / Verschmelzung

——— Der folgende Text ist ein direktes Zitat von [22]. ———

m eine Truppe zu verstärken, kann der Gruppenmanager anordnen, dass mehrere weitere Soldaten zu einer bestehenden Gruppe hinzugefügt werden. Dabei geht die Authorisierte Instanz (AI) in CAKE äquivalent wie beim Einzeleintritt vor [9]. Jedem neuen Mitglied wird das neue Gruppenschlüsselpaar aus GTEKneu und GKEKneu verschlüsselt mit seinem persönlichen Schlüssel gesendet. Die restlichen Teilnehmer erhalten das Datenpaket mit den neuen Schlüsselinformationen, die mit dem bisherigen GKEKaktuell kodiert sind. Auf diese Weise werden pro neuem Mitglied zwei Nachrichten aus Sicht des GCKS verschickt.
Alternativ lassen sich die Schlüsselpaare aus Primzahl und geheimen Schlüssel aller neuen Teilnehmer sowie die der bestehenden Mitglieder über ein CRT zusammenfassen. Der GKEKneu wird dabei jeweils mit den privaten KEKs verschlüsselt und gemeinsam mit den Primzahlen entsteht das Secure Lock. Das Datagramm kann anschließend mit dem mit GKEKneu überschlüsseltem GTEKneu addiert werden und per Broadcast gesendet werden. So wäre sowohl für den GCKS als auch für alle Clients jeweils nur eine einzige Nachricht vonnöten.

——— Ende des Zitats. ———

Mehrfach-Austritt / Aufteilung

——— Der folgende Text ist ein direktes Zitat von [22]. ———

benso kann vom Gruppenmanager angeordnet werden, mehrere Soldaten zugleich aus einer Gruppe, die verkleinert werden soll, zu entnehmen, um ihnen jeweils separate Aufgaben zuzuweisen. Dabei unterscheidet sich der Algorithmus, den der GCKS verrichten muss, kaum vom Einzelaustritt. Hierzu werden in der ternären Baumstruktur entsprechend mehrere Pfade markiert. Der restliche Vorgang geschieht wie beim Einzelaustritt in 4.3.3.

——— Ende des Zitats. ———

4.3.5 Re-Key einer Gruppe

——— Der folgende Text ist ein direktes Zitat von [22]. ———

ür das Re-Keying des GTEK einer Gruppe existieren zwei Möglichkeiten. In der ersten Variante generiert der GCKS oder ein beliebiger Gruppenteilnehmer ein neues Gruppenschlüsselpaar $GTEK_{neu}$ und $GKEK_{neu}$, welches mit dem noch verwendeten $GKEK_{aktuell}$ verschlüsselt wird. Das somit entstandene Datagramm wird im Anschluss per Broadcast an alle Gruppenteilnehmer verschickt.

Durch das Festlegen eines klar definierten Zeitpunktes können die Gruppenmitglieder synchron auf den neuen $GTEK_{neu}$ umstellen.

Datagram a: mit $GKEK_{aktuell}$ verschlüsseltes Schlüsselpaar $GTEK_{neu}$,$GKEK_{neu}$
Datagram b: $GKEK_{neu}$SL und $GTEK_{neu}$ XOR $GKEK_{neu}$

Die zweite Option sieht die Nutzung der individuellen Schlüssel der Teilnehmer bzw. des CRT Systems vor. Dabei verfährt der GCKS entsprechend des Konzeptes der initialen Erzeugung einer Gruppe bzw. des ersten GTEK.
In der Implementierung dieser Arbeit wurde die erste Variante übernommen, da dies weniger Rechenaufwand für den GCKS darstellt. Im Zusammenhang mit dem gestellten Szenario muss die Rechenpower möglichst effizient genutzt werden. Die Erstellung des neuen Schlüsselpaares und die anschließende Überschlüsselung mit dem $GKEK_{aktuell}$ ist nicht so rechenintensiv wie der Aufwand des CRTs. Bei der Wahl der zweiten Möglichkeit müsste neben der Erstellung des neues Schlüsselpaares die Primzahlen der Gruppenteilnehmer miteinander multipliziert werden. Ebenso muss im Kontext der CRT Berechnung der $GKEK_{neu}$ mit den jeweiligen persönlichen KEKs der Gruppenteilnehmer verschlüsselt und miteinander multipliziert werden. Mit weiteren Multiplikationen und Modulo Rechnungen entsteht letztendlich das Secure Lock. Wie in der Abbildung zu sehen ist, muss der $GKEK_{neu}$ mit dem $GTEK_{neu}$ XOR berechnet werden und zum Datagram hinzugefügt werden.

Allerding ist bei beiden Varianten der Nachrichtenaufwand der Gleiche. Es ist jeweils eine Broadcast notwending. Empfängt ein Client Datagram a, so entschlüsselt er das Paket mit dem in seinem Besitz vorhandenen $GKEK_{aktuell}$. Empfängt er Datagram b, entschlüsselt er das Secure Lock, indem er darauf die Modulo Funktion mit seinem persönlichen KEK anwendet. So erhält er den $GKEK_{neu}$ und damit kann der $GTEK_{neu}$ geparst werden, indem die XOR Funktion auf den zweiten Teil des Datagramms angewendet wird Aus Sicht der Gruppenmitglieder stellt die Wahl der Datagramme zum Einen in der Anzahl der empfangenen Nachrichten keinen Unterschied. Jeder empfängt ein Rekey Paket. Der Rechenaufwand, die dem Client zum $GTEK_{neu}$ führen, ist bei der zweiten Variante gering höher. Für den GCKS ist wie beschrieben die Wahl der ersten Variante deutlich vorteilhafter.

——— Ende des Zitats. ———

5 Implementierung

Für die Implementierung des Protokolls wurde das in Kapitel 2.2 beschriebene Betriebssystem RIOT gewählt. Als Programmiersprache dient C. Anders als vom Design vorgesehen, werden bei der Implementierung die kryptographischen Algorithmen nicht von Client und GCKS ausgehandelt, sondern im Vorfeld definiert. Dies hat den Hintergrund, dass die Berechnung und Verteilung der Gruppenschlüssel anhand des *Chinese Remainder Theorems* Hauptmerkmal dieser Arbeit ist.

5.1 Datenstrukturen

Für die Funktionsweise des Protokolls ist die Definition einiger Datenstrukturen notwendig, welche beispielsweise das Abbilden von Clients und Gruppen erlauben.

5.1.1 Nachrichten

Die vom GNRC Network Stack verwendeten, und im *pktbuf* zwischengespeicherten Nachrichten bestehen aus einer verketteten Liste. Die Elemente dieser Liste bestehen aus Paket „Stückchen“ (*packet snips*) und werden durch das von RIOT definierte Struct *gnrc_pktsnip_t* abgebildet. Jedes dieser Structs stellt entweder einen Header oder eine Nutzlast dar. Abbildung 5.5 stellt diesen Aufbau am Beispiel eines UDP Pakets dar.

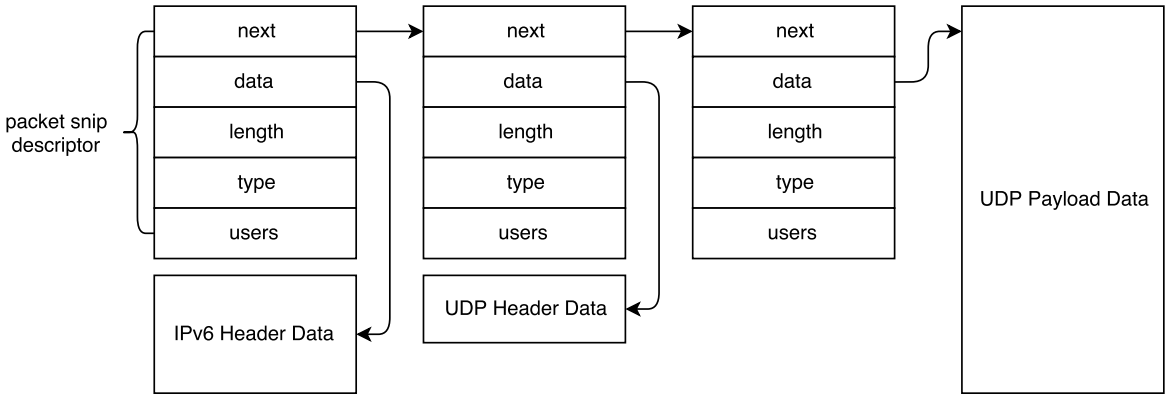

Abbildung 5.1: Aufbau eines GNRC Pakets

Das Hinzufügen eines *gnrc_pktsnip_t* in den *pktbuf* erfolgt über den Aufruf der Funktion *gnrc_pktbuf_add()*. Diese erwartet das nächste Element des Pakets, einen Pointer auf die Daten, die Länge der Daten und den Protokolltyp als Parameter und gibt einen Pointer auf die Speicheradresse im *pktbuf* zurück. Hierbei wird das erste ausreichend große, freie Segment im Puffer ausgewählt, wodurch durchaus eine Fragmentierung des *pktbuf* entstehen kann. Die

empfangenen Pakete bestehen aus einer verketteten Liste mit nur einem Element, welches die gesamten Daten beinhaltet. Durch den Aufruf Funktion *gnrc_pktbuf_mark()* kann dieses Element geteilt werden, wobei die ersten n Bytes des Pakets als nächstes Element an das ursprüngliche Element angehängt wird. Dadurch kann das Paket für die weitere Verarbeitung in die einzelnen Header und Payloads aufgetrennt werden. Wurde eine Nachricht bearbeitet oder wird sie nicht länger benötigt, kann der im *pktbuf* belegte Speicher mithilfe der Funktion *gnrc_pktbuf_release()* wieder freigegeben werden.

5.1.2 Ternäre Baumstruktur

——— Der folgende Text ist ein direktes Zitat von [22]. ———

ie Verschlüsselung und Verteilung der Gruppenschlüssel bei der initialen Erzeugung einer Gruppe und bei der Austrittoperation verlangt die Berechnung des CRT, in der unter anderem der GKEK verschlüsselt mit dem privaten KEK jedes Clients beim anfänglichen CRT bzw. beim Austritt mit jeweils verschiedenen KEKs, die in den entsprechenden Knoten liegen, beinhaltet sind.

Der Schlüsselbaum, dessen Knoten jeweils eine Primzahl und ein KEK beinhalten, kann von der Wurzel aus nach unten aufgebaut und gefüllt werden.

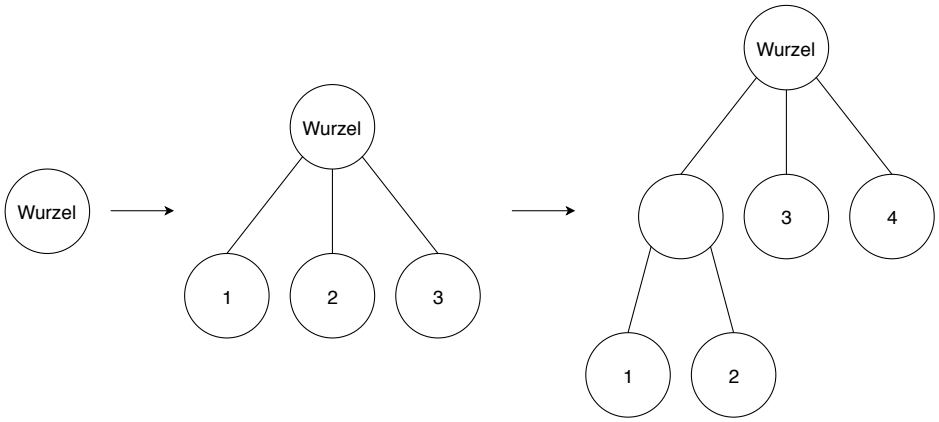

Abbildung 5.2: Baum 1-1

Für jedes Gruppenmitglied wird ein neuer Blattknoten eingefügt, dessen Primzahl und persönlicher KEK im entsprechenden Blattknoten beinhaltet sind.

Ausgehend von der Wurzel können also die ersten drei Blattknoten angehängt werden (Abbildung 5.2). Kommt ein vierter Client hinzu, müsste einer der drei Clients zu einem Zwischenknoten umgewandelt werden, bestehend aus einer neuen Primzahl und einem KEK. An dem neuen Zwischenknoten kann dann der Clientknoten, dessen Position der neue Zwischenknoten eingenommen hat, kann der alte dem der Clientknoten, dessen Platz der neue Knoten einnimmt, als neue Kindknoten angehängt wird. Auf diese Weise ändert sich der Vaterknoten des Clients mit der Nummer 1 (Abbildung **??**). Nachdem die zwei anderen Clients 2 und 3 ebenfalls zu Zwischenknoten umgewandelt wurden und jeweils drei Clients an allen drei Zwischenknoten eingefügt wurden, müsste beim nächsten Einfügen wieder einer der Clients aus der untersten Ebene zu einem Zwischenknoten umgewandelt werden und das gleiche Prozedere wie beschrieben durchgehen. Der Nachteil einer solchen Einfügemethode führt dazu, dass die Clients bei Join Operationen neue Vaterknoten erhalten würden und

diese noch einmal einzeln angesprochen werden müssten, um ihnen ihre aktuellen Schlüsseln mitzuteilen.

Eine idealere Baumstruktur wäre, das Paradigma des B+-Baumes zu übernehmen. Beim Einfügen wird nicht in die Tiefe eingesetzt, sondern es wird bei einem Overflow in einer Ebene eine neue Wurzel angelegt (Abbildung 5.4).

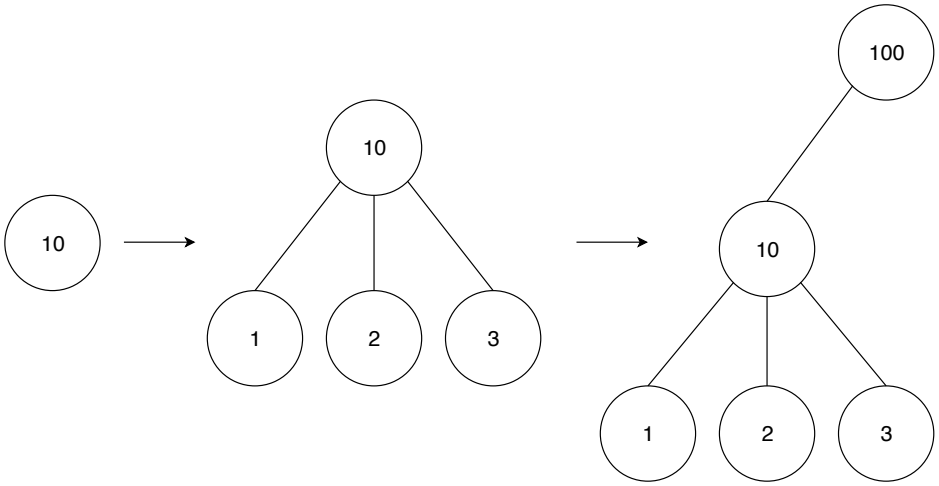

Abbildung 5.3: Baum 2-1

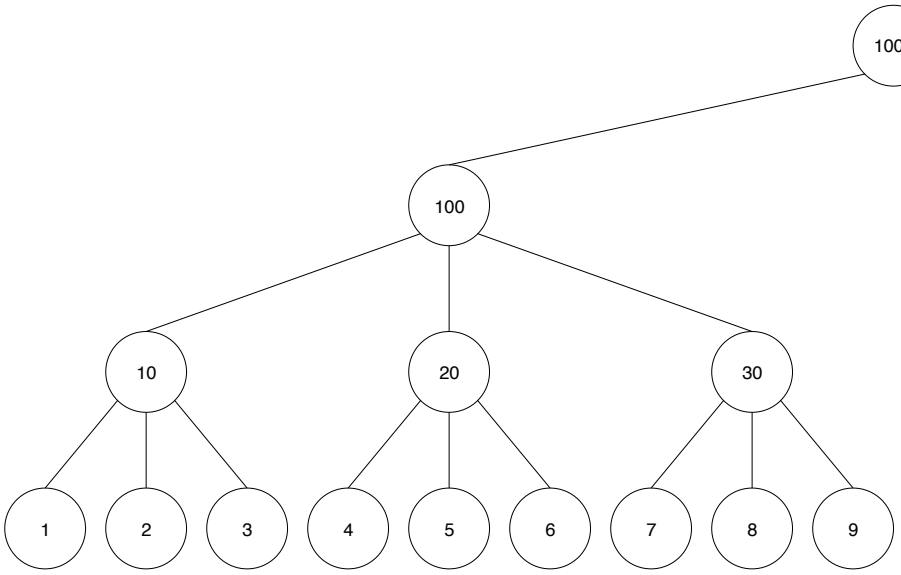

Abbildung 5.4: Baum 2-2

Sobald eine neue Wurzel erstellt wurde, sollten aufgrund der gleichen Problematik wie oben beschrieben, die zwei restlichen freien Plätze der Wurzel nicht für Kindknoten besetzt werden, sondern es werden von der Wurzel aus so lange neue Zwischenknoten angelegt, bis Zwischenknoten mit Level 1 erstellt wurden. An diesen können dann Kindknoten eingefügt werden.

Dadurch, dass sich bei einem Overflow einer Ebene eine neue Wurzel angelegt wird, ist es möglich, allen Teilnehmern dieses Teilbaumes über eine Broadcast über die Wurzel zu

informieren. Der Wort Case, jeden Teilnehmer einzeln per Unicast anzusprechen, tritt beim ersten Austritt auf. Bei der ersten Leave-Operation einer Gruppe haben die Gruppenteilnehmer nämlich keinerlei Kenntnis über ihre Zwischenknoten und ihren Pfad bis zur Wurzel.

In der Implementierung dieser Arbeit wird bei der Austritt Operation der Fokus auf den Worst Case gelegt. Den Gruppenteilnehmern wird einzeln per Unicast mitgeteilt, mit welchem Knoten sie das CRT auflösen sollen und dadurch den GKEK erhalten können.

——— Ende des Zitats. ———

5.2 Konfiguration

——— Der folgende Text ist ein direktes Zitat von [22]. ———

ie Quelldatei kann vor dem Kompilieren und Ausführen über eine Konfigurationsdatei angepasst und dabei verschiedene Parameter verändert werden. So kann zum einen das aus einer Zeichenkette bestehende PSK in eine beliebig andere verändert werden. Zum anderen lassen sich Parameter wie die maximale Anzahl der Clients, die eine Gruppe aufnehmen kann, eingestellt werden. Die Anzahl der Gruppen selbst lässt sich ebenfalls regulieren.
Die Portnummer kann über die Variable `CAKE_PORT` gesetzt werden. Wie bereits in Kapteil 4 beschrieben, wird standardmäßig die Portnummer 849 verwendet.

Außerdem können die Längen veschiedener Schlüssel eingestellt werden.

- Die Bytelänge der im CAKE System notwendigen Schlüssel wie GTEK, GKEK und KEK können mit `DEF_KEY_LENGTH` gesetzt werden.
- Die Länge der für jeden Client zugewiesenen und für jeden Baumknoten erstellte Primzahl kann mit `PRIME_LENGTH` verändert werden. Hierbei sollte beachtet werden, dass die Primzahl stets größer sein sollte als die in den CRT gelangte KEK Variable, damit die Berechnungen des CRT (siehe Kapitel 5.5) funktionieren. Standardmäßig hat `PRIME_LENGTH` den Wert `DEF_KEY_LENGTH + 1`.

——— Ende des Zitats. ———

Nachrichten werden über den Aufruf der *cake_receive()* Funktion empfangen. Da diese Funktion das Programm solange blockiert, bis eine Nachricht empfangen wurde, muss diese in einen zweiten Thread ausgelagert werden, um weiterhin eine Kommunikation zwischen Client und GCKS zu ermöglichen.

```
1 static msg_t _rcv_msg_queue[MAIN_QUEUE_SIZE];
2 char rcv_thread_stack[THREAD_STACKSIZE_MAIN];
3
4 void *rcv_thread(void *arg)
5 {
6     (void) arg;
7
8     msg_init_queue(_rcv_msg_queue, MAIN_QUEUE_SIZE);
9
```

```
10      gnrc_netreg_entry_t receive_netreg = GNRC_NETREG_ENTRY_INIT_PID(
11          GNRC_NETREG_DEMUX_CTX_ALL, sched_active_pid);
12      receive_netreg.demux_ctx = CAKE_PORT;
13      gnrc_netreg_register(GNRC_NETTYPE_UDP, &receive_netreg);
14
15      while (1) {
16          gnrc_pktsnip_t *msg = cake_receive();
17          msg_handle(msg);
18          gnrc_pktbuf_release(msg);
19      }
20
21      return NULL;
22  }
23
24  thread_create(rcv_thread_stack, sizeof(rcv_thread_stack),
25                THREAD_PRIORITY_MAIN - 1, THREAD_CREATE_STACKTEST,
26                rcv_thread, NULL, "rcv_thread");
```

Code 5.1: Erstellung eines zweiten Threads

Um Nachrichten empfangen zu können, müssen die jeweiligen Threads zuerst einen entsprechenden Eintrag im Netzwerk Register (*netreg*) hinterlegen. Dazu wird die Thread ID sowie ein *demux*-Kontext benötigt. Über den *demux*-Kontext lassen sich Pakete des selben Typs weiter unterscheiden. Da das Protokoll auf UDP basiert, ist der *demux*-Kontext die Portnummer.

```
1  client_info_t clients[MAX_CLIENT_COUNT] = {{0}};
2  group_info_t groups[MAX_GROUP_COUNT] = {{0}};
```

Code 5.2: Initialisierung der Client- und Gruppen-Arrays

Der für die Client und Gruppen Strukturen benötigte Speicher wird in Abhängigkeit der Konstanten *MAX_CLIENT_COUNT* und *MAX_GROUP_COUNT* alloziiert.

5.3 Nachrichtenverarbeitung

Um die in diesem Protokoll versendeten Nachrichten korrekt zuzuordnen und verarbeiten zu können, wird zunächst ein weiteres Struct definiert, durch das ein eigener Headertyp abgebildet wird. Das Feld *exchange_type* gibt hierbei an, um welchen Nachrichtentyp des Protokolls es sich handelt.

```
1  typedef struct cake_hdr {
2      uint8_t exchange_type;
3      payload_type_t pl_next;
4      network_uint16_t msg_id;
5      network_uint16_t length;
6  } __attribute__ ((packed)) cake_hdr_t;
```

Code 5.3: CAKE Header Struct

Beim Empfangen einer Nachricht wird zunächst überprüft, welchen Typ diese Nachricht hat, damit entsprechend darauf reagiert werden kann. Dies erfolgt über die *msg_handle* Funktion. Ein Auschnitt der Funktion wird in Code 5.4 gezeigt.

```
uint8_t msg_handle(gnrc_pktsnip_t *msg)
{
    size_t len = msg->size;
    cake_hdr_t *hdr = cake_hdr_handle(msg);

    if (hdr == NULL) {
        DEBUG("ERROR: Could not parse CAKE header");
        return EXIT_ERROR;
    }

    if (byteorder_ntohs(hdr->length) != len) {
        DEBUG("ERROR: Invalid packet length");
        return EXIT_ERROR;
    }

    if (hdr->exchange_type == CAKE_INIT && client.state == STATE_IDLE) {
        if (!handle_init_response(msg)) {
            return EXIT_ERROR;
        }
    }
    else if (hdr->exchange_type == CAKE_AUTH && client.state == STATE_INIT) {
        if (!handle_auth_response(msg)) {
            return EXIT_ERROR;
        }
    }
    else if (hdr->exchange_type == CAKE_GROUP_JOIN &&
             client.state == STATE_AUTH) {
        if (!handle_join_group_response(msg)) {
            return EXIT_ERROR;
        }
    }

    // [...]

    else {
        DEBUG("ERROR: Unknown Exchange Type - aborting!");
        return EXIT_ERROR;
    }

    return EXIT_SUCCESS;
}
```

Code 5.4: *msg_handle* Funktion des Clients

Wurde ein gültiger Nachrichten Typ ermittelt, ruft der Client die entsprechende *handle*-Funktion auf. Der GCKS verhält sich beim Empfangen einer Nachricht analog dazu.

An das *packet snip*, das den CAKE Header enthält, werden nacheinander die einzelnen Payloads angehangen. Aufgrund der Tatsache, dass die Nachrichten als ein einziges Datagramm empfangen werden, bedarf es eine Lösung, wie man die Daten wieder auftrennen kann. Jeder

Payload wird daher mit einem eigenen Header versehen, der die Länge der Daten, sowie den Typ des nächsten Payloads enthält.

```
typedef struct payload_hdr {
    payload_type_t pl_next;
    network_uint16_t length;
} __attribute__ ((packed)) payload_hdr_t;
```

Code 5.5: Payload Header Struct

5.4 Initialisierung und Authentifizierung

Der Client wird mittels der *init_client_info()* initialisiert. Hierbei wird unter die IPv6 Adresse sowie eine eindeutige, auf der Seriennummer der CPU basierende, ID ausgelesen. Zusätzlich dazu werden die für den Diffie-Hellman-Schlüsselaustausch benötigten Werte generiert.

```
client_info_t client_info;

uint8_t init_client_info(void)
{
    client_info.state = STATE_IDLE;
    client_info.id = get_luid();
    client_info.addr = get_ipv6_from_ifs();
    client_info.dh_ctxt = init_dh_ctxt();
}
```

Code 5.6: Initialisierung des Clients

```
dh_context_t *init_dh_ctxt(void)
{
    dh_ctxt.curve = uECC_secp256r1();
    dh_ctxt.public_size = uECC_curve_public_key_size(dh_ctxt.curve);
    dh_ctxt.private_size = uECC_curve_private_key_size(dh_ctxt.curve);
    dh_ctxt.compressed_public_size = dh_ctxt.private_size + 1;

    if (!uECC_make_key(dh_ctxt.public, dh_ctxt.private, dh_ctxt.curve)) {
        return NULL;
    }

    uECC_compress(dh_ctxt.public, dh_ctxt.compressed_public, dh_ctxt.curve);

    return &dh_ctxt;
}
```

Code 5.7: Generierung der Diffie-Hellman-Werte

Die Registrierung des Clients am GCKS erfolgt über den Aufruf der zwei Funktionen *cake_init()*, welche den Diffie-Hellman-Schlüsselaustausch implementiert, und *cake_auth()*, wobei sich Client und GCKS gegenseitig mittels des PSK authentifizieren.

5.5 Berechnung des CRT

Da bei der Berechnung des CRTs, in Abhängigkeit von der Schlüssellänge des GKEKs, der Länge der Primzahlen, sowie der Anzahl an Clients über die das CRT erstellt wird, mitunter sehr große Zahlen entstehen können, Standard C aber keinen Datentyp für Zahlen enthält, der mehr als 64 Bit darstellen kann, muss eine externe Bibliothek eingesetzt werden. Hierbei handelt es sich die GNU Multiple Precision Arithmetic Library (GMP), beziehungsweise eine Teilmenge dieser Bibliothek, die durch die Datei *mini-gmp.c* bereitgestellt wird. GMP wird in Kapitel 5.8 tiefergehend behandelt.

Wie vom Protokoll festgelegt, wird zunächst der GKEK der Gruppe mit dem KEK eines jeden Clients verschlüsselt. Der verschlüsselte GKEK und die Primzahl des Clients werden im Anschluss in einen GMP Integer umgewandelt.

```
mpz_t primes[n];
mpz_t encr_gkeks[n];

uint8_t encr_gkek[DEF_KEY_LENGTH] = {0};

for (int i = 0; i < n; i++) {
    // Initialisierung der mpz_t Variablen
    mpz_init(primes[i]);
    mpz_init(encr_gkeks[i]);

    // XOR-Verschlüsselung des GKEK mit dem KEK eines Clients
    xor_encr(gkek, DEF_KEY_LENGTH, encr_gkek, clients[i]->keymat.key);

    // Einlesen und Umwandeln des verschlüsselten GKEKs in einen GMP Integer
    mpz_import(encr_gkeks[i], DEF_KEY_LENGTH, 1, 1, 0, 0, encr_gkek);

    // Einlesen und Umwandeln der Primzahl des Clients in einen GMP Integer
    mpz_import(primes[i], PRIME_LENGTH, 1, 1, 0, 0, clients[i]->prime);
}
```

Code 5.8: Umwandlung der GKEKs und Primzahlen

Das CRT kann nun mittels der Funktion *chinese_remainder()* berechnet werden. Als Parameter werden ihr der Pointer des Speicherbereichs übergeben, in das das Ergebnis geschrieben werden soll, jeweils ein Pointer auf die Arrays der verschlüsselten GKEKs und Primzahlen, sowie die Anzahl der Elemente, über die das CRT berechnet werden soll.

```
void chinese_remainder(mpz_t *crt, mpz_t encr_gkeks[], mpz_t primes[], size_t
    n)
{
  mpz_t prod, quot, sum, tmp;

  // Initialisierung der mpz_t Variablen
  mpz_init_set_ui(prod, 1);
  mpz_init_set_ui(sum, 0);
  mpz_init(quot);
```

```
9     mpz_init(tmp);
10
11    // Berechnung des Produkts über alle Primzahlen
12    mpz_mul_arr(prod, primes, n);
13
14    for (size_t i = 0; i < n; i++) {
15      mpz_cdiv_q(quot, prod, primes[i]);
16      mpz_invert(tmp, quot, primes[i]);
17      mpz_mul(tmp, tmp, encr_gkeks[i]);
18      mpz_addmul(sum, tmp, quot);
19    }
20
21    mpz_mod(sum, sum, prod);
22
23    mpz_set(*crt, sum);
24
25    // Freigabe des belegten Speichers
26    mpz_clear(prod);
27    mpz_clear(quot);
28    mpz_clear(sum);
29    mpz_clear(tmp);
30  }
```

Code 5.9: *chinese_remainder()* Funktion

Die hierbei entstehende Zahl, das SL, wird im Anschluss als Zeichenkette abgelegt, damit es an die Clients verschickt werden kann. Hierzu wird zunächst über den Aufruf der Funktion *mpz_sizeinbase()* die Anzahl der Stellen des SL ermittelt und ein Byte-Array in der entsprechende Größe initialisiert. Hierbei muss beachtet werden, dass zu dem Rückgabewert der *mpz_sizeinbase()* Funktion der Wert 2 hinzu addiert werden muss, da die Funktion *mpz_get_str()* zwei Elemente des Arrays für ein potentielles Minuszeichen sowie den Nullterminator beansprucht (Code 5.10).

```
1 size_t crt_length = mpz_sizeinbase(crt, 10) + 2;
2
3 uint8_t crt_str[crt_length];
4 mpz_get_str((char *) crt_str, 10, crt);
```

Code 5.10: Umwandlung des CRTs in eine Zeichenkette

Diese wandeln die Zeichenkette wieder in einen GMP Integer um und führen die Umkehrfunktion aus. Das Ergebnis dieser Operation ist der mit dem privaten Schlüssel des Clients verschlüsselte GKEK. Dieser wird vom Client entschlüsselt (Code 5.11, woraufhin er sich im Besitz des Gruppenschlüssels, dem GTEK, befindet.

```
1 mpz_t crt, prime, tmp_gkek;
2
3 mpz_init(crt);
4 mpz_init(prime);
5 mpz_init(tmp_gkek);
6
7 // Einlesen des erhaltenen Strings
```

```
8  mpz_set_str(crt, crt_str, 10);
9
10 // Einlesen und Umwandeln der Primzahl des Clients in einen GMP Integer
11 mpz_import(prime, PRIME_LENGTH, 1, 1, 0, 0, client_prime);
12
13 // Modulo Berechnung
14 mpz_mod(tmp_gkek, crt, prime);
15
16 // Exportieren als Byte-Array
17 mpz_export(gkek, NULL, 1, 1, 0, 0, tmp_gkek);
18
19 mpz_clear(tmp_gkek);
20 mpz_clear(prime);
21 mpz_clear(crt);
22
23 // Entschlüsselung des GKEKs mit dem KEK
24 xor_encr(gkek, DEF_KEY_LENGTH, gkek, client_kek);
```

Code 5.11: Umwandlung und Lösen des CRTs, Entschlüsselung des GKEK

5.6 Gruppenoperationen

Neben der Mlglichkeit eine neue Gruppe anzufordern, unterstützt die Implementierung einige der in Kapitel 4 beschriebenen Gruppenoperationen. Diese sind der Einzeleintritt, der Einzelaustritt, sowie das Re-Keying. Hierzu schickt der Client dem GCKS einen Payload mit der der entsprechenden Gruppen ID, auf die die Operation angewandt werden soll. Durch die Gruppen ID kann der GCKS die Operation eindeutig einer Gruppe zuordnen kann. Die Art der auszuführenden Operation wird im CAKE Header angegeben.

Erstellen einer Gruppe

Über die Funktion *cake_create_group()* kann ein Client den GCKS dazu auffordern eine neue Gruppe zu erstellen. In der Funktion *new_group()* kontrolliert der GCKS zunächst ob sich in dem Gruppen-Array noch ein freies Element befindet. Ist dies der Fall, erstellt er einen zufälligen GKEK und GTEK, generiert eine ID für die Gruppe und überprüft diese auf Einzigartigkeit.

```
1  group_info_t *new_group(group_info_t *groups)
2  {
3      group_info_t *group = find_free_group(groups);
4
5      if (group == NULL) {
6          return NULL;
7      }
8
9      generate_keypair(group->keys.gtek, group->keys.gkek);
10
11     uint8_t gid;
12     do {
13         cake_random(&gid, sizeof(gid));
14     } while (!gid_is_unique(gid, groups));
```

```
15
16      group->id = gid;
17
18      return group;
19  }
```

Code 5.12: Erstellen einer neuen Gruppe mittels *new_group()*

Im weiteren Verlauf des Programms wird dem Client diese ID in einem entsprechendem Payload zurückgesendet. Der Client trägt die ID seinerseits in seinem Gruppen-Array ein.

Einzeleintritt

Für den Gruppenbeitritt steht dem Client die Funktion *cake_join_group()* zur Verfügung. Als Parameter wird die ID der Gruppe, der beigetreten werden soll, übergeben. Nach einem erfolgreichen Beitritt wartet der Client darauf, dass der GCKS ihm das Schlüsselmaterial zusendet.

Der GCKS überprüft zunächst ob eine Gruppe mit entsprechender ID existiert und fügt den Client hinzu, falls dies der Fall ist. Der weitere Programmablauf hängt in erster Linie davon ab, ob die Gruppenschlüssel bereits erstmalig verteilt wurden.

Ist dies nicht der Fall, muss der GCKS überprüfen ob bereits ausreichende Teilnehmer vorhanden sind. Über die Konfigurationsdatei kann festelegt werden, ab dem wievielten Client das CRT für die erste Schlüsselverteilung berechnet wird. Wurde diese Anzahl durch den Eintritt des Clients nicht erreicht, wartet der GCKS auf weitere Teilnehmer. Wurde die Mindestanzahl an Gruppenmitgliedern hingegen erreicht, berechnet der GCKS das CRT nach dem in Kapitel 5.5 beschriebenen Verfahren und verteilt es zusammen mit dem durch den GKEK verschlüsselten GTEK.

Wurde hingegen bereits Schlüsselmaterial verteilt, wird kein CRT berechnet, sondern ein neues, zufälliges Schlüsselpaar generiert. Die bereits vorhandenen Teilnehmer der Gruppe erhalten das neue Schlüsselpaar mittels einer Broadcast Nachricht, die die mit dem bereits bekannten GKEK verschlüsselten neuen Schlüssel enthält als Payload enthält. Der beigetretene Client erhält das neue Schlüsselpaar vie Unicast. Dieses wird zuvor mit dem seinem privaten Schlüssel (KEK) verschlüsselt.

```
1  uint8_t gkek_old[DEF_KEY_LENGTH];
2  memcpy(gkek_old, group->keys.gkek, DEF_KEY_LENGTH);
3
4  generate_keypair(group->keys.gtek, group->keys.gkek);
5
6  uint8_t encr_keys[2 * DEF_KEY_LENGTH];
7  xor_encr(group->keys.gtek, 2 * DEF_KEY_LENGTH, encr_keys, gkek_old);
8
9  send_keypair_multicast(group, encr_keys);
10 send_keypair_to_client(client->addr, &group->keys);
```

Code 5.13: Schlüsselverteilung

Einzelaustritt

——— Der folgende Text ist ein direktes Zitat von [22]. ———

...]

——— Ende des Zitats. ———

Re-Key

Da nur der Group Manager dazu berechtigt ist, den Re-Key Prozess manuell einzuleiten, überprüft der GCKS beim Empfang einer entsprechenden Nachricht, ob es sich bei dem Absender um den Group Manager handelt (Code **??**). Der Re-Key Vorgang kann außerdem nur eingeleitet werden, wenn die ersten Schlüssel bereits verteilt wurden.

```
1 if (group && group->manager->id == client->id &&
2           group->initial_keys_distributed) {
3
4     // Schlüsselerstellung und -verteilung
5
6 }
```

Code 5.14: Überprüfung der Berechtigung zum Re-Key

Die Erzeugung neuer Schlüssel erfolgt wie bei dem oben beschriebenen Einzeleintritt, mit dem Unterschied, dass nur eine Nachricht mittels Broadcast an die Gruppe verschickt wird.

5.7 Verschlüsselung

RIOT OS bietet zum Entstehungszeitpunkt dieser Arbeit keine Möglichkeit Daten eines sich im *pktbuf* befindlichen Pakets effizient zu verschlüsseln. Die Problemstellung wird in [8] behandelt und durch die Einführung der *gnrc_pktbuf_merge()* Funktion gelöst. Dabei werden die Daten einer verketteten Liste von *snips* in ein einzelnes, neu angelegtes *snip* geschrieben. Aus dem so enstandenen Speichersegment können im Anschluss die Daten für die Verschlüsselung ausgelesen werden.

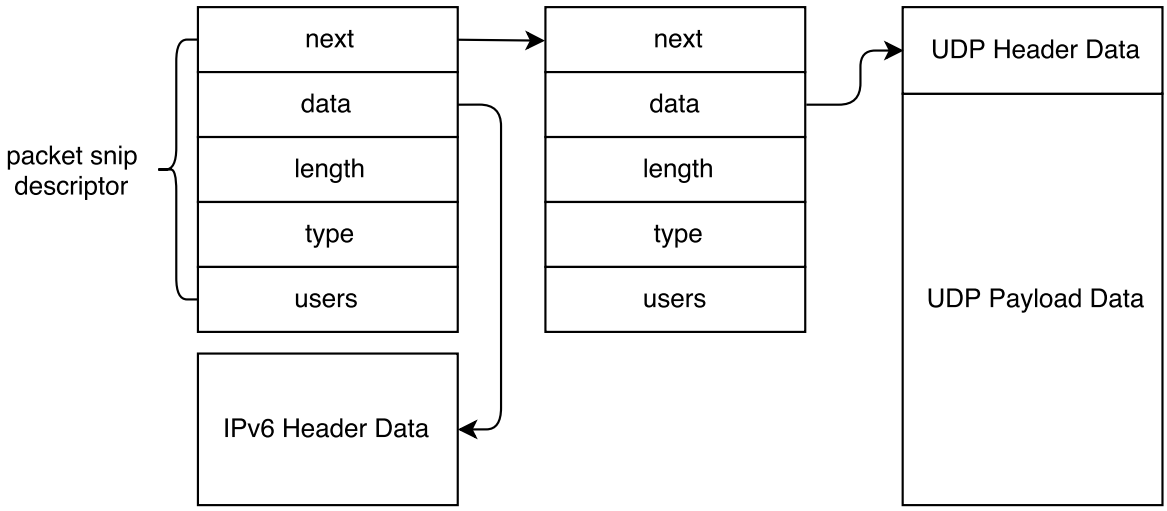

Abbildung 5.5: UDP Paket nach Anwendung der *gnrc_pktbuf_merge()* Funktion

5.8 Verwendete Module und Bibliotheken

Die folgenden Module und Bibliotheken stellen die vom Protokoll benötigten und eingesetzten kryptographischen und arithmetischen Operationen zur Verfügung.

RIOT Module Besonders nennenswert sind die zwei RIOT Module *Hashes* [18] und *Crypto* [17]. Beide lassen sich über das *Makefile* des Programms einbinden und stellen die eingesetzten kryptographischen Hashverfahren sowie Verschlüsselungsmethoden zur Verfügung.

GNU Multiple Precision Arithmetic Library [5] Für die Berechnungen des CRTs werden Zahlentypen mit größerer Genauigkeit benötigt, als von Standard C unterstützt werden. Da der Großteil der durch GMP bereitgestellten Funktionen in dieser Implementierung keine Verwendung findet, wurde stattdessen die kleinere Bibliothek *mini-gmp* benutzt. Diese stellt ebenfalls alle für die Arithmetik großer Zahlen notwendigen Funktionen bereit.

micro-ecc [15] RIOT stellt keine Funktionen für einen Diffie-Hellman-Schlüsselaustausch zur Verfügung. Die *micro-ecc* Bibliothek kann allerdings über das von RIOT zur Verfügung gestellte *package interface* eingebunden werden. *micro-ecc* stellt eine Reihe von elliptischen Kurven zur Verfügung, unter anderem *secp256r1*, welche standardmäßig in der Implementierung verwendet wird.

6 Evaluation

Das in Kapitel 4 spezifizierte und in Kapitel 5 umgesetzte Protokoll wurde gegen die in Kapitel 3 festgelegten Anforderungen getestet.

6.1 Testumgebung und Szenario

Als Testumgebung dient der von RIOT OS zur Verfügung gestellte *native port*. Dieser ermöglicht das Ausführen von RIOT OS als Prozess in einem *NIX System, ohne dabei auf die eigentliche Hardware angewiesen zu sein. Diese ist in Abbildung 6.1 dargestellt. Auf der rechten Seite wird das Serverprogramm ausgeführt, auf der linken Seite insgesamt drei Clients. Durch die in Kapitel 6.2 beschriebenen Probleme verliefen Tests auf mobilen Endgeräten bislang nicht erfolgreich. Da die CPU-Leistung im *native port* nicht künstlich gedrosselt werden kann, ist eine qualitative Aussage über die Geschwindigkeit der Implementierung zu diesem Zeitpunkt nicht möglich. Die Ergebnisse aus [8] können als Referenzwerte hergenommen werden. Der darin entwickelte Client dient der Implementierung teilweise als Grundlage.

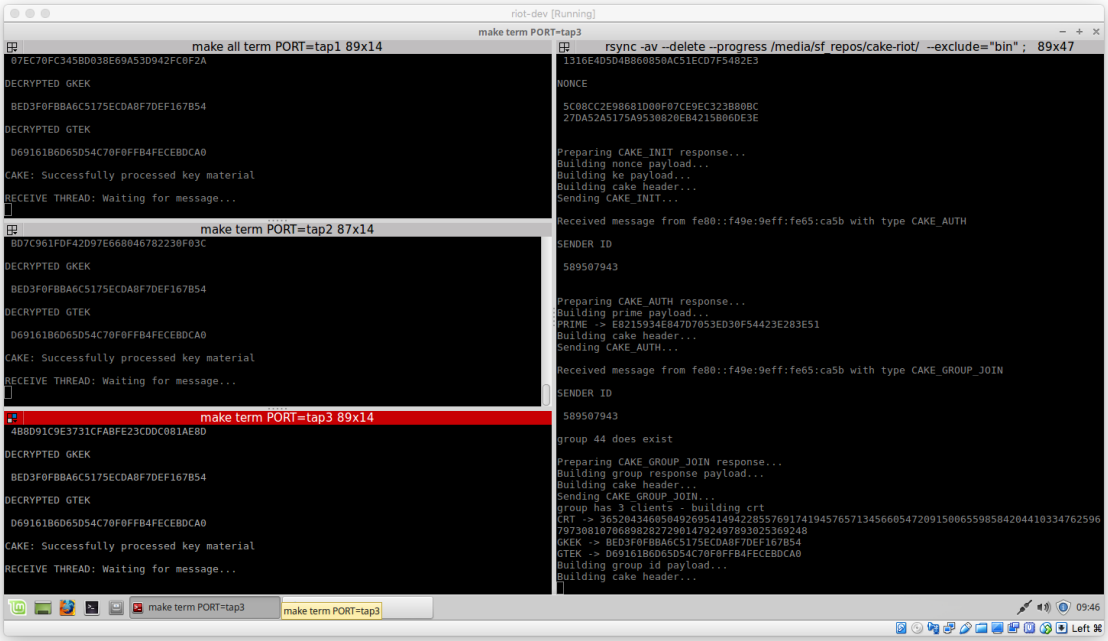

Abbildung 6.1: Testumgebung

Das in Kapitel 2.1 beschriebene Szenario wurde manuell getestet.

6.2 Bewertung

CAKE und das darum liegende Protokoll, erfüllen alle in 3.2.1 beschriebenen Anforderungen. Unautorisierte Nutzer, die in den Besitz von Nachrichten gelangen, können diese nicht entschlüsseln, da sie nicht über den dafür notwendigen GTEK verfügen. Ebensowenig können sie ein abgefangenes CRT lösen, wenn sie sich nicht unter den dafür bestimmten Empfängern befinden, da ihr privater Schlüssel und Primzahl nicht in die Berechnung mit eingeflossen ist. Diese Eigenschaft gewährleistet auch die Forward Secrecy. Verlässt ein Teilnehmer eine Gruppe oder wird er von dieser ausgeschlossen, so folgt zwangsläufig die Berechnung eines neuen CRTs, welches vom ausgeschlossenen Teilnehmer nicht gelöst werden kann. Der entstehende Rechenaufwand wird hierbei minimiert, da das CRT nicht über alle Knoten des ternären Baumes berechnet werden muss, sondern nur über einen Teilbaum, in dem sich der ehemalige Teilnehmer befand. Die Schlüsselunabhängikeit ist dadurch gewährleistet, dass bei Verschlüsselung des GTEKs mit dem GKEK eine XOR-Verknüpfung angewandt wird, welche ohne Kenntnis über den eigentlichen Schlüssel nachweislich nicht zu dechiffrieren ist [13].

6.2.1 Speicherbedarf

Die Datenstrukturen des System wurden weitestgehends so definiert, dass sie auf dynamische Speicherzuweisung verzichten. Bei der Berechnung des CRTs hat sich dieser Versuch als problematisch erwiesen, da der benötigte Speicher von der Anzahl der Clients abhängt, und somit erst zur Laufzeit ermittelt werden kann. Dem *mpz_t* Struct, durch welches die Primzahlen und KEKs bei der Berechnung dargestellt werden, kann zwar ein statischer Bereich im Speicher zugewiesen werden, dieser müsste aber so groß sein, dass das Produkt der Primzahlen aller n Clients einer Gruppe abgebildet werden kann. Bei einer maximalen Anzahl von 81 Clients, und einer Primzahlengröße von 32 Byte, entspräche dies im schlimmsten Fall einem Speicherbereich von mindestens $121 * 32\,\text{Byte} = 3872\,\text{Byte}$, da hier noch interne Pointer des *mpz_t* Structs dazugerechnet werden müssen.

6.2.2 Anzahl der versendeten Nachrichten

——— Der folgende Text ist ein direktes Zitat von [22]. ———

...]

——— Ende des Zitats. ———

7 Zusammenfassung und Ausblick

——— Der folgende Text ist ein direktes Zitat von [22]. ———

n diesem Kapitel werden Aspekte angeprochen, die in weiteren Arbeiten rund um das CAKE Protokoll hinzugenommen werden können.

Tritt ein neuer Teilnehmer einer gesicherten Gruppenkommunikation bei, so sieht CAKE vor, eine Unicast an den neuen Teilnehmer zu senden und eine Multicast an die restlichen Teilnehmer der Gruppe. Der Nachrichtenaufwand aus Sicht des GCKS beträgt dabei somit zwei. Erst bei einem Masseneintritt wird die Berechnung des CRT als Alternative für das äquivalente Vorgehen beim Einzeleintritt vorgeschlagen [9] (siehe auch 4.3.2 und 4.3.3). Die Anzahl der zu versendenen Nachrichten vom GCKS könnte im Algorithmus des Einzeleintritts auf eins verringert werden, indem wie bei der initialen Erzeugung der Gruppe und der Leave Operation ein CRT berechnet wird. Dabei fließt die Primzahl und sein geheimer Schlüssel des neuen Teilnehmers in die Berechungen des CRT gemeinsam mit den Schlüsselpaaren der anderen Gruppenteilnehmer mit ein. Der entstandene Secure Lock kann anschließend per Multicast verschickt werden. Auf diese Weise reduziert sich die zu versendene Nachricht auf eins und die Backward Secrecy bleibt erhalten. Im Zusammenhang des beschriebenen Szenarios ist die Funkübertragung teuer und sollte so gering wie möglich gehalten werden. Dadurch wäre diese Methode vorteilhafter. Auf der anderen Seite muss der dadurch verbundene erhöhte Rechenaufwand berücksichtigt werden, der für den GCKS entsteht. Hier wird deutlich, dass der Vorteil der Nachrichtenanzahl zum Nachteil des Rechenpowers für den GCKS führt.

Möchte ein Gruppenteilnehmer T1 einem anderen Gruppenteilnehmer T2 etwas mitteilen, verschlüsselt er die Nachricht N mit dem GTEKaktuell und versendet diese. Kommt ein Rekey R1 zustande und wird dadurch der Gruppenschlüssel aktualisiert, bevor Teilnehmer T2 die Nachricht N empfängt, so kann er die Mitteilung nicht entschlüsseln, weil der Gruppenschlüssel aktualisiert wurde. Dies kann so gelöst werden, indem mithilfe des NTP Protokolls in jeder Nachricht ein Zeitstempel hinzugefügt wird und die Gruppenteilnehmer mindestens die letzten zwei Gruppenschlüssel abspeichern. So kann zumindest der passende Gruppenschlüssel für die Nachrichten ausgewählt werden, die vor dem Rekey R1 und nach dem Rekey R1 erstellt und versendet wurden. In zukünftigen und detaillierteren Arbeiten kann das eventuell berücksichtigt werden.

Die initiale Erzeugung einer gesicherten Gruppenkommunikation und die Beitrittsoperation benötigen keinerlei Verwaltungsmanagement der in 4.3.3 beschriebenen Baumstruktur. Auch die in diesen zwei Gruppenoperationen notwendigen Datenpakete, welche vom GCKS zu den entsprechenden autorisierten Clients gelangen, kommen ohne den ternären Baum aus. Zwingend notwendig ist die Verwaltung des Schlüsselbaumes aber im Zusammenhang eines Austritts, da der verkleinerte CRT, der berechnet wird, die Schlüsselpaare aus KEK und Primzahl aus den entsprechenden Baumknoten benötigt. Überlässt man die Arbeiten an der Baumstruktur nur in der Leave Operation, entsteht ein sehr hoher Nachrichtenaufwand bei jeder Austrittsoperation. So würde bei der ersten Leave Operation der Worst Case

eintreffen. Der Worce Case ist, von Seiten des GCKS n Nachrichten verschicken zu müssen, wobei n die Anzahl der restlichen Gruppenteilnehmer nach einem Austritt beschreibt. Das ist deshalb der Fall, weil die Gruppenmitglieder keine Kenntnisse über ihren Pfad bis zur Wurzel haben. Die Arbeiten um den Schlüsselbaum könnten teilweise in die Beitrittsoperation verlagert werden, indem den entsprechenden Teilnehmer nähere Informationen über ihren Pfad bis zur Wurzel mitgeteilt wird.
Zusätzlich kann im Schlüsselbaum eine Semantik entworfen werden, die den Gruppenteilnehmern ermöglicht, den Knoten, den sie zum Auflösen des CRT benötigen, selbst zu errechnen, nachdem sie aufgrund eines Austritts ein neues Datenpaket erhalten haben, das in Form von dem aufzubrechenden Secure Lock den neuen Gruppenschlüssel enthält.

——— Ende des Zitats. ———

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] Emmanuel Baccelli u. a. „RIOT: One OS to rule them all in the IoT“. Diss. INRIA, 2012.

[2] M. Baugher u. a. *The Internet Key Exchange (IKE)*. RFC 4046. Apr. 2005. URL: https://rfc-editor.org/rfc/rfc4046.txt.

[3] Guang-Huei Chiou und Wen-Tsuen Chen. „Secure broadcasting using the secure lock“. In: *IEEE Transactions on Software Engineering* 15.8 (Aug. 1989), S. 929–934. ISSN: 0098-5589. DOI: 10.1109/32.31350.

[4] A. Dunkels, B. Gronvall und T. Voigt. „Contiki - a lightweight and flexible operating system for tiny networked sensors“. In: *29th Annual IEEE International Conference on Local Computer Networks*. Nov. 2004, S. 455–462. DOI: 10.1109/LCN.2004.38.

[5] Torbjörn Granlund. *GNU MP*. Version 6.0.0. Free Software Foundation. 2014.

[6] T. Hardjono, S. Rowles und B. Weis. *The Group Domain of Interpretation*. RFC 6407. Okt. 2011. URL: https://rfc-editor.org/rfc/rfc6407.txt.

[7] D. Harkins und D. Carrel. *The Internet Key Exchange (IKE)*. RFC 2409. Nov. 1998. URL: https://rfc-editor.org/rfc/rfc2409.txt.

[8] Tobias Heider. „Minimal G-IKEv2 implementation for RIOT OS“. Bachelor's Thesis. Ludiwg-Maximilians-Universität München, Apr. 2017.

[9] Peter Hillmann, Marcus Knüpfer und Gabi Dreo Rodosek. „CAKE: Hybrides Gruppen-Schlüssel-Management Verfahren“. In: *10. DFN-Forum Kommunikationstechnologien*. Hrsg. von Paul Müller u. a. Bonn: Gesellschaft für Informatik e.V., 2017, S. 31–40.

[10] C. Kaufman u. a. *Internet Key Exchange Protocol Version 2 (IKEv2)*. STD 79. http://www.rfc-editor.org/rfc/rfc7296.txt. RFC Editor, Okt. 2014. URL: http://www.rfc-editor.org/rfc/rfc7296.txt.

[11] Martine Lenders. „Analysis and Comparison of Embedded Network Stacks“. Master's Thesis. Freie Universität Berlin, Apr. 2016.

[12] P. A. Levis. „TinyOS: An Open Operating System for Wireless Sensor Networks (Invited Seminar)“. In: *7th International Conference on Mobile Data Management (MDM'06)*. Mai 2006, S. 63–63. DOI: 10.1109/MDM.2006.151.

[13] C. Matt und U. Maurer. „The one-time pad revisited“. In: *2013 IEEE International Symposium on Information Theory*. Juli 2013, S. 2706–2710. DOI: 10.1109/ISIT.2013.6620718.

[14] D. Maughan u. a. *Internet Security Association and Key Management Protocol (ISAKMP)*. RFC 2408. Nov. 1998. URL: https://rfc-editor.org/rfc/rfc2408.txt.

[15] *micro-ecc Webseite*. URL: http://kmackay.ca/micro-ecc/ (besucht am 05. 04. 2018).

[16] Sandro Rafaeli und David Hutchison. „A Survey of Key Management for Secure Group Communication“. In: 35 (Sep. 2003), S. 309–329.

[17] *RIOT Crypto Modul*. URL: http://riot-os.org/api/group__sys__crypto.html (besucht am 05. 04. 2018).

[18] *RIOT Hashes Module*. URL: http://riot-os.org/api/group__sys__hashes.html (besucht am 05. 04. 2018).

[19] Bundesamt für Sicherheit in der Informationstechnik. *BSI-Grundschutz Katalog*. 1996. URL: http://www.bsi.de/gshb/deutsch/index.htm.

[20] Brian Weis, Yoav Nir und Valery Smyslov. *Group Key Management using IKEv2*. Internet-Draft draft-yeung-g-ikev2-13. http://www.ietf.org/internet-drafts/draft-yeung-g-ikev2-13.txt. IETF Secretariat, März 2018. URL: http://www.ietf.org/internet-drafts/draft-yeung-g-ikev2-13.txt.

[21] H. Will, K. Schleiser und J. Schiller. „A real-time kernel for wireless sensor networks employed in rescue scenarios“. In: *2009 IEEE 34th Conference on Local Computer Networks*. Okt. 2009, S. 834–841. DOI: 10.1109/LCN.2009.5355049.

[22] Mehdi Yosofie. „CAKE - Hybrides Gruppenschlüsselmanagementprotokoll für RIOT OS“. Bachelor's Thesis. Ludiwg-Maximilians-Universität München, Apr. 2018.